

Ein universelles Programmsystem
für den Einsatz eines Rechners zur
Experimentsteuerung und Datenaufnahme
über CAMAC

von
Tassilo Schlurick
aus Halle/Saale

1976

Ein universelles Programmsystem für den Einsatz eines Rechners zur Experimentsteuerung und Datenaufnahme über CAMAC

Den Naturwissenschaftlichen Fachbereichen
der Friedrich-Alexander-Universität Erlangen-Nürnberg

zur

Erlangung des Doktorgrades

vorgelegt von

Tassilo Schlurick

aus Halle/Saale

Als Dissertation genehmigt von den Naturwissenschaftlichen
Fachbereichen der Universität Erlangen-Nürnberg.

Tag der mündlichen Prüfung: 24.11.1976.

Vorsitzender der Promotionskommission:	Professor Dr. E. Schweizer
Erstberichterstatte:	Professor Dr. N. Fiebiger
Zweitberichterstatte:	Professor Dr. F. Hofmann

Inhaltsverzeichnis

Seite

1.	Einführung	1
1.1	Der Rechner im Experiment	1
1.2	Einige Gesichtspunkte zur Entwicklung der Experimentelektronik	3
1.3	Zusammenfassung der wesentlichen Merkmale von CAMAC	5
1.4	Der CAMAC-Kontroller an der PDP 7	7
2.	Beurteilung verfügbarer Software	11
2.1	Assembler- und Macrosprachen	11
2.2	ALGOL und FORTRAN	11
2.3	Realzeitsprachen	12
2.4	Interpretative Sprachen	13
2.5	Die CAMAC-Sprachen	14
2.6	Zusammenfassende Beurteilung	14
3.	Das CAMAC-orientierte Analysier- und Auswertesystem (COAAS)	16
3.1	Zusammenstellung der Grundzüge	16
3.2	Struktur des Systems	17
3.2.1	Simulation der Handbedienung	17
3.2.2	Die Taskerstellung	19
3.2.2.1	Die OPEN-Anweisung	19
3.2.2.2	Die TASK-Anweisung	20
3.2.2.3	Die END- und RETURN-Anweisungen	20
3.2.2.4	Starten einer Task	21
3.2.2.5	Beispiel einer Task	22
4.	Sprachzusammenstellung	23
4.1	Zahlen und Adreßdarstellungen	23
4.1.1	Konstanten	23
4.1.1.1	Ganzzahlige Konstanten	23
4.1.1.1.1	Integerkonstanten	23
4.1.1.1.2	Oktalkonstanten	23
4.1.1.1.3	CAMAC-Adreßkonstanten	23
4.1.1.2	Realzahlkonstanten	24
4.1.2	Variablen	24
4.1.2.1	Einfache Variablen	25
4.1.2.1.1	Realzahl- und Integervariablen	25
4.1.2.1.2	CAMAC-Adreßvariablen	25
4.1.2.2	Datenfelder	25
4.1.2.3	Datenbereiche der Spektren	25
4.2	Arithmetische Ausdrücke	26
4.3	Laufanweisung	26
4.4	Sprunganweisungen	26
4.4.1	Unbedingte Sprunganweisung	26
4.4.2	Bedingte Sprunganweisung	27
4.5	Standard Ein-Ausgabe-Anweisungen	27
4.6	Die FORMAT-Anweisung	27
4.7	Anweisungsnummern	28
4.8	Die Leeranweisung	28
4.9	Kommentaranweisung	28

4.10	CAMAC-Anweisungen	28
4.10.1	Allgemeine Beschreibung	28
4.10.2	Unterteilung in Gruppen	29
4.10.2.1	Anweisungen der ersten Gruppe	29
4.10.2.2	Anweisungen der zweiten Gruppe	30
4.10.2.3	Anweisungen der dritten Gruppe	30
4.10.3	Behandlung der ADC's in CAMAC	31
4.10.4	Das 24 Bit-Datenwort in CAMAC und die Q-Behandlung	32
4.10.4.1	Schreiben der obersten 6 Bit	32
4.10.4.2	Lesen der obersten 6 Bit	32
4.10.4.3	Die Q-Abfrage	32
4.11	Übergeordnete Anweisungen	33
4.11.1	Fileeröffnungsanweisung	33
4.11.2	Löschen eines Files	33
4.11.3	Löschen des Directorys	33
4.11.4	Errichtung einer Task	33
4.11.5	Beendigung einer Task	35
4.11.6	Die TAKE-Anweisung	35
4.11.7	Der Anschluß einer Task an CAMAC	36
4.11.8	Der Start einer Task	36
4.12	Zusammenfassung der Analysier- und Auswertebefehle	36
4.12.1	Bildschirmbefehle	37
4.12.2	Analysierbefehle	37
4.12.3	DECTapebefehle	38
4.12.4	Arbeitsbefehle	38
4.12.5	Konfigurationsbefehle	39
5.	Struktur des Übersetzers	40
6.	Das Betriebssystem	43
6.0	Struktur des Betriebssystems	43
6.1	Der Taskkontroller	47
6.2	Der Startmechanismus	49
6.3	Der Taskwechsel	50
6.4	Das Taskzustandsdiagramm	51
6.5	Die CAMAC-Interruptverarbeitung	53
6.6	Der CAMAC-Device-Driver	54
6.7	Die Kernspeicherbelegung	56
6.8	Der Anschluß der FORTRAN-Unterprogramme	57
6.9	Dynamische Kenngrößen des Systems	59
6.10	Fehlerbehandlung	62
7.	Der Einsatz des Systems im Experiment	64
7.1	Die Meßanordnung	64
7.2	Das Meßprogramm	67
7.3	Messungen	68
8.	Schlußbemerkung	71
	Literaturverzeichnis	72
	Anhang A: Programm zur Messung von T_C -Änderungen	A1
	Anhang B: Programm zum Übertrag von Spektren	B1
	Anhang C: CAMAC-Macros	C1
	Anhang D: Betriebsanleitung	D1
	Anhang E: Aufbau des Compilers	E1

Zusammenfassung

Zur Steuerung von Experimenten und zur Aufnahme von Meßdaten mit einer Rechenanlage PDP 7 über einen CAMAC-Kontroller CCC 2 wurde ein Programmsystem CoAAS entwickelt, das vom Experimentator in einer höheren Programmiersprache programmiert werden kann. Der Kern der Sprache besteht aus einer "vereinfachten FORTRAN-Sprache", die für den Realzeitbetrieb mit einer Taskstruktur erweitert wurde. Für die Ansteuerung von CAMAC existieren spezielle Anweisungen, die auch direkt vom Fernschreiber ausgeführt werden können. Ein Realzeitbetriebssystem steuert die Bearbeitung von Task's nach LAM's von CAMAC. An einem Beispiel ist der Einsatz im Experiment demonstriert.



Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41546221_0077

1. Einführung

1.1 Rechner und Experiment

Der Einsatz von Prozeßrechnern in der experimentellen Kernphysik stellt besondere Anforderungen an die Hardwareausrüstung der Rechner und damit verbunden auch an die Software. Obwohl die on-line-Datenverarbeitung und das rechnergesteuerte Experimentieren zur Realzeitdatenverarbeitung gehören, und deshalb viele Probleme mit den Aufgaben in der industriellen Prozeßsteuerung vergleichbar sind, gibt es eine Reihe von Besonderheiten. Begründet sind diese in der Art der Anwendung, der Benutzung eines Rechners zur Durchführung eines Experiments.

Schon aus der wörtlichen Bedeutung von Experiment lassen sich die besonderen Schwierigkeiten für die Hard- und Softwareentwicklung ableiten. Während die Automatisierung eines industriellen Prozesses ein einmaliger Vorgang ist, der nach entsprechender Systemanalyse konkret geplant und dann durch eine z. T. auch speziell entwickelte Betriebssoftware über Prozeßsteuerungselemente, die am Rechner angeschlossen sind und in den Prozeß eingreifen, realisiert wird, wird zum Experimentieren ein wandelbarer Versuchsaufbau mit einem anpassungsfähigen Interface zum Rechner benötigt. Da immer wieder neue Experimente kreiert werden, und die Ergebnisse dieser Experimente völlig neue Erkenntnisse zeitigen sollen, müssen die Aufbauten und auch die Versuchsbedingungen solange verändert werden, bis die gesuchten Informationen anfallen. Bei der Durchführung eines Experiments spielt oft eine systematische Variation von Parametern bei fest gegebenem Anfangszustand eine wesentliche Rolle.

Neben der geforderten hohen Flexibilität kommt noch erschwerend hinzu, daß der Datenanfall stochastisch ist und oft kritische Zeitbedingungen einzuhalten sind. Dabei kann sogar nur selten die Zeitauflösung vom Rechner selbst vorgenommen werden, und es ist in der Regel eine zusätzliche Experimentlogik für die Zeitbestimmung erforderlich. Der Rechner muß aber in der Lage sein, hohe Datenraten verarbeiten zu können. Eine wesentliche Aufgabe der Experimentsteuerung kann z. B. sein, Veränderungen in der Experimentlogik zur Zeitbestimmung vorzunehmen.

Die Zeitspanne für den Aufbau und die Durchführung eines Versuchs schwankt zwischen einigen Stunden und einigen Jahren. Eventuell ist es noch notwendig bei sich widersprechenden Ergebnissen, Messungen zu wiederholen, aber meist ist mit der Erlangung des Meßergebnisses der Experimentaufbau überflüssig geworden und es kann ein neues Experiment in Angriff genommen werden. Das schließt aber nicht aus, daß Teile eines Experimentaufbaus auch für andere Versuche eingesetzt werden können, allerdings wird ein neues Experiment oft unter ganz anderen Gesichtspunkten gesehen. Daraus ist ersichtlich, daß auch eine hohe Anpassungsfähigkeit über das einzelne Experiment hinaus ein wesentliches Ziel der Experimentiertechnik sein muß.

1.2 Einige Gesichtspunkte zur Entwicklung der Experimentelelektronik

Verfolgt man die Entwicklung der Experimentelelektronik, so kann man feststellen, daß die ersten elektronischen Entwicklungen speziell für jeden Versuch vorgenommen wurden. Es stellte sich jedoch bald heraus, daß diese allerdings auch an Umfang zunehmenden Aufbauten in logische Bausteine gegliedert werden können, und daß die Bausteine wiederum unabhängig voneinander produziert werden können, wenn sie an den Schnittstellen angepaßt sind. Als Folge dieser Erkenntnis wurde das NIM-System genormt. In diesem System, das analoge Signale verarbeitet, ist es möglich, elektronische Geräte verschiedener Hersteller zu kombinieren und auch Eigenentwicklungen durchzuführen. Das NIM-System ist weltweit verbreitet, und es wird sicherlich noch längere Zeit ohne wesentliche Änderungen der Spezifikation, aber mit einer wachsenden Vielfalt von Bausteinen in der experimentellen Kernphysik, aber auch in anderen Bereichen z.B. der Medizin eingesetzt.

Eine ähnliche Entwicklung machten die Datenerfassungssysteme durch. Auf diesem Sektor gab es lange Zeit und gibt es auch noch heute Spezialentwicklungen. Besonders zu erwähnen ist die Entwicklung von Vielkanalanalysatoren. Auch hier zeigte sich, daß standardisierte universelle Geräte flexibler und z.T. auch billiger sind. Deshalb hat sich der Einsatz von Rechnern, speziell Prozeßrechnern, für die Vielkanalanalyse durchgesetzt. Zu Beginn wurden sie nur zur Datenaquisition benutzt und hier optimal den Aufgaben angepaßt.

Es wurde aber bald die Frage gestellt, inwieweit die Fähigkeiten eines Rechners weiter ausgenutzt werden können, indem der Rechner auch in das Experiment eingreift, um z.B. stumpfsinnige, oft wiederkehrende oder aber auch komplizierte Schaltmechanismen, die z.T. sehr fehleranfällig von der Bedienung her sind, zu übernehmen.

Für diese Aufgaben wurden für verschiedene Experimente Einzellösungen verwirklicht, die Spezialinterface an den Rechnern benötigten, die wiederum oft nur für ein Experiment, aber sicherlich nur für einen Rechner-typ verwendet werden konnten.

Lösungen dieser Art sind vergleichbar mit den Aufgaben der Prozeßautomatisierung. Sie benötigen, weil verschiedene Probleme wie z.B. Geräteentwicklung, Prozeß-rechneranpassung, Interfaceentwicklung und Software-erstellung zu bewältigen sind, einen großen personellen Aufwand. Der Experimentator kann die Probleme ohne Spezialisten nicht mehr lösen. Es gibt zwar noch die Möglichkeit, Interface der verwendeten Rechner zu benutzen, die für die Prozeßautomatisation entwickelt wurden, aber die Anpassungen an die Experimentelektronik bleiben trotzdem ungelöst.

Um einen Rechner für die Experimentautomatisierung einsetzen zu können, müssen deshalb zwei Voraussetzungen erfüllt sein:

1. Dem Experimentator muß eine Palette von logischen und funktionellen Bausteinen ähnlich dem NIM-System zur Verfügung gestellt werden, die für seine Aufgaben geeignet sind, und die er unkompliziert an den Rechner anschließen kann. Hierzu sind auch reine Anpassungsbausteine zu zählen, die die Verbindung zu anderen Geräten übernehmen.
2. Der Experimentator benötigt eine Software, mit der er diese Bausteine bedienen und in der er seine Probleme formulieren und lösen kann. Diese Software muß für den Experimentator durchschaubar sein.

Während die erste Voraussetzung durch die Spezifikation des CAMAC-Systems /1,2/ bereits erfüllt ist, befaßt sich diese Arbeit mit der Lösung des zweiten Problems. Bevor die Softwareseite der Experimentautomatisierung behandelt wird, sollen kurz die wesentlichen Merkmale des CAMAC-Systems zusammengestellt werden.

1.3 Zusammenfassung der wesentlichen Merkmale von CAMAC

Das CAMAC-System wurde für den Einsatz von Prozeßrechnern in der experimentellen Kern- und Hochenergiephysik erstellt. In der vorliegenden Arbeit werden die Spezifikationen von CAMAC vorausgesetzt.

Zusammenfassende und Anwenderliteratur findet der Leser in /3,4,5/. An dieser Stelle sollen nur die wesentlichen Merkmale von CAMAC und seine Folgerungen für den Experimentator zusammengefaßt werden.

1. Computerunabhängigkeit

Die wichtigste Eigenschaft von CAMAC ist seine Computerunabhängigkeit. Bei der Entwicklung von Experimentelektronik brauchen keine speziellen Eigenschaften eines Rechners berücksichtigt zu werden. Bei Einhaltung der Spezifikationen kann jedes CAMAC-Gerät an jedem mit einem CAMAC-Interface ausgestatteten Rechner arbeiten.

2. Lösung des Interface-Problems

Mit dem Anschluß eines CAMAC-Kontrollers an einen Rechner ist prinzipiell das Interfaceproblem Rechner - Experimentelektronik gelöst. Das bedeutet allerdings nicht, daß damit auch jede benötigte Übertragungsleistung zur Verfügung steht. Vor der Entwicklung eines Kontrollers müssen immer noch Untersuchungen über den Leistungsbedarf der Experimentelektronik angestellt werden, und der Controller muß daraufhin abgestimmt werden.

3. Modularer Aufbau

CAMAC ist ein modulares System. Dadurch kann das Gerätespektrum leicht auf den speziellen Bedarf abgestimmt werden, ohne daß unnötige Funktionsbausteine, wie sie bei kompakten Systemen oft auftreten, berücksichtigt werden müssen. Außerdem kann der Experimentator erwarten, da das System auch rechnerunabhängig ist, daß eine große Anzahl verschiedenster Funktionsmodule bereits kommerziell gefertigt wird. Es entfallen dadurch langwierige und kostspielige Entwicklungsarbeiten für den Experimentator.

4. Mehrstufige Adressierung

Durch die Verwendung einer mehrstufigen Adressierung können der Moduladreibereich und der Umfang der benötigten adressierbaren Modulplätze unabhängig voneinander realisiert werden. Für jeden Modul, der im allgemeinen einem Gerät entspricht, steht ein eigener Adreibereich zur Verfügung, der entsprechend der Anzahl der benötigten Datenregister ausgebaut werden kann. Unabhängig davon hat der Benutzer die Möglichkeit, die Größe seines CAMAC-Interfaces durch die Anzahl der Crates und Branches zu bestimmen, in denen die Module untergebracht sind.

5. Abstrakter Funktions- und Datencode

In CAMAC ist ein abstrakter Funktions- und Datencode festgelegt. Für den Anwender ist nur die Kenntnis des genormten Codes sowie die Abbildung der Funktionen eines jeden Moduls auf diesen Code erforderlich. Diese Zuordnungen sind aus den Modulbeschreibungen der Module ersichtlich.

6. Synchronisation

Jeder Modul kann über eine Steuerleitung (L) bei Bedarf eine Anforderung LAM (Look at me) an den Rechner stellen, damit dieser die Bedienung des Moduls oder einer ganzen Modulgruppe vornimmt. Diese Anforderung schafft die Grundlage für die Realzeitverarbeitung im CAMAC-System.

7. Sondersignale

Zwei zusätzliche Leitungen (X und Q) ermöglichen einen flexiblen Betrieb von CAMAC. Während die Signalleitung X zur automatischen Fehlererkennung dient, kann Q in verschiedenen Varianten für eine Leistungssteigerung des Kontrollers verwendet werden. So können z.B. mit dem Signal Q verschiedene automatische Übertragungsarten zwischen einem externen Gerät (über ein CAMAC-Modul) und dem Rechner realisiert werden.

8. Gemeinsame Synchronisation

Zur gemeinsamen Synchronisation aller CAMAC-Module innerhalb des Interface stehen Steuerleitungen zur Verfügung. Damit können CAMAC-Module gleichzeitig gesperrt (I), entsperrt, gelöscht (C) und initialisiert (Z) werden.

1.4 Der CAMAC-Kontroller an der PDP 7

Der an der PDP 7 angeschlossene CAMAC-Kontroller ist ein Einzelcratekontroller, der von der Österreichischen Studiengruppe für Atomenergie für den Rechner PDP 8 entwickelt wurde /6/. Aufgrund der Tatsache, daß der Interruptmechanismus und die Zeitablaufsteuerung der Ein-Ausgabe-Befehle (IOT's) bei beiden Rechnern gleich strukturiert sind, konnte der Kontroller mit entsprechender Anpassungsschaltung auch für die PDP 7 verwendet werden /7/.

Der Controller arbeitet im Einzelworttransfer. Die Abb. 1 zeigt ein Blockschaltbild. Der Befehls- und Datenverkehr wird über 4 Register abgewickelt, die über den AC geladen werden. Die beiden 12 Bit Datenregister können außerdem in den AC gelesen werden. Während das Adreßregister nur die CAMAC-Adresse CNA aufnimmt, enthält das Funktionsregister zusätzlich zu den CAMAC-Funktionen (F) noch weitere Crate-Funktionen für die Steuerung des Controllers. Die Tabelle 1 gibt einen Überblick über die Crate-Funktionen.

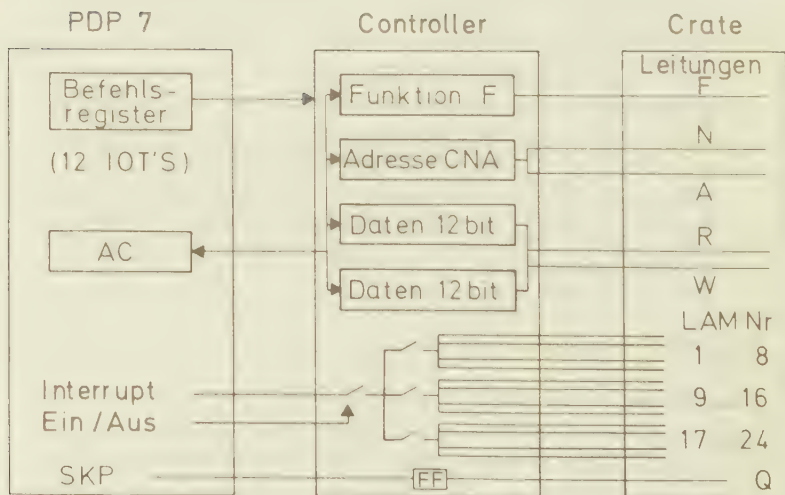


Abb. 1: Blockschaltbild des CAMAC-Kontrollers an der PDP 7

Bedeutung	Crate-Funktion
CAMAC Steuerbefehle	CF=0
Clear Crate	CF=1
Initialize	CF=2
Set Inhibit	CF=3
Clear Inhibit	CF=4
Read LAM	CF=5
Read	CF=6
Write	CF=7
Transfer Write	CF=8

Tab. 1: Crate-Funktionen

Zur Bedienung des Kontrollers werden 8 IOT-Befehle benutzt, die in Tabelle 2 zusammengestellt sind. Zur Ausführung eines CAMAC-Befehls müssen in der Regel 2 IOT's für Steuerbefehle und 4 IOT's für Datentransfers gegeben werden. Der Start eines CAMAC-Zyklus kann, da die IOT-Befehle mikroprogrammierbar sind, gleichzeitig mit einem der Registerladebefehle erfolgen.

Bedeutung	Befehlscode	IOT-Befehle
Skip if LAM	CCSIL	701601
Skip if LAM im adressierten Crate	CCSIC	701621
CAMAC-Disable	CCDIS	701641
Skip if Q	CCSIQ	701661
CAMAC-Enable	CCENA	701645
Lade Adreßregister	CCDAA	701602
Lade Funktionsregister	CCDAF	701622
Lade Bit 13-24	CCDAU	701642
Lade Bit 1-12	CCDAL	701662
Lese Bit 13-24	CCLAU	701652
Lese Bit 1-12	CCLAL	701672
Starte CAMAC-Zyklus	CCSTR	7016X4

Tab. 2: IOT-Befehle

Die 22 L-Leitungen von den Modulplätzen werden in 3 Gruppen zu einem Interruptsignal verodert. Jede Gruppe kann über einen Schalter am Kontroller abgeschaltet werden, so daß zu Prüfzwecken oder zur erhöhten Sicherheit von abgeschalteten Gruppen kein Interrupt verursacht werden kann. Der CAMAC-Interrupt kann außerdem softwaremäßig vom Rechner geschaltet werden.

Über eine Crate-Funktion kann das L-Muster in die Datenregister übertragen und von dort durch normale Leseoperationen zur Verarbeitung in den AC gebracht werden. Das L-Muster ist nach den Modulplätzen geordnet und die Priorität kann nur softwaremäßig entschieden werden.

Wie bereits in der Zusammenfassung der Merkmale von CAMAC hingewiesen wurde, hängt die Leistungsfähigkeit der Übertragungen vom Kontroller ab. Durch den Einzelworttransfer ist der Einsatz des Kontrollers für datenintensive Übertragungen weniger geeignet, da diese sinnvollerweise über Datenkanäle abgewickelt werden sollten. Deshalb liegt auch der Haupteinsatz von CAMAC an der PDP 7 auf dem Gebiet der Experimentsteuerung, zumal die datenintensiven Analog- zu -Digital-Konverter mit einem eigenen Interface angeschlossen sind. Der Einsatz von ADC's in CAMAC sollte nur bei komplizierten Experimenten erfolgen, bei denen wiederum keine zeitkritischen Bedingungen zu erwarten sind.

Das X-Signal wurde im Crate-Kontroller der PDP 7 noch nicht implementiert, weil es erst später in die Spezifikationen von CAMAC aufgenommen wurde.

2. Beurteilung verfügbarer Software

Bevor der eigene Lösungsvorschlag behandelt wird, soll betrachtet werden, wie bestehende Software zur Lösung der CAMAC-Programmierung eingesetzt werden könnte. Dabei soll keine Rücksicht genommen werden, ob diese Software für den Rechner verfügbar ist oder nicht.

Zur Beurteilung reicht nicht die isolierte Betrachtung der Programmiersprache aus, sondern einen wesentlichen Anteil für ihren Einsatz hat auch das Betriebssystem, das einen Realzeitbetrieb ermöglichen muß.

2.1 Assembler- und Macrosprachen

Assembler- und Macrosprachen sind aufgrund ihrer Maschinenbezogenheit am besten geeignet, die wirkungsvollsten und an die jeweilige Rechanlage angepaßtesten CAMAC-Programme zu entwickeln. Die Einführung von Macros für CAMAC setzt voraus, daß entsprechende Routinen für die CAMAC-Bedienung entwickelt und ins Betriebssystem integriert werden. Ein Vorteil bei der Verwendung von Assemblersprachen liegt auch in der Möglichkeit, in begrenzten Fällen einfachere Betriebssysteme zu verwenden, bei denen die Realzeitbedingungen noch vom Benutzer bearbeitet werden. Es ist allerdings nicht mehr zumutbar, daß die Assemblerprogramme vom Experimentator geschrieben und ausgetestet werden. Deshalb kommt ihre Verwendung nur für Demonstrationsexperimente und bei großen Experimenten in Frage, die einen entsprechenden Personalaufwand erlauben.

2.2 ALGOL und FORTRAN

Obwohl ALGOL und FORTRAN keine Realzeitsprachen sind, und sich deshalb die Aufgaben der Experimentautomatisierung schlecht formulieren lassen, haben diese Sprachen den Vorteil, daß sie weit verbreitet sind, und die meisten Experimentatoren eine von beiden beherrschen. Deshalb kommt auch der Entwicklung von

REAL-TIME-FORTRAN eine große Bedeutung zu. Die wesentlichen Koordinierungsaufgaben werden dabei vom Betriebssystem übernommen, und der Benutzer bearbeitet über ein Unterprogrammsystem die Prozeß- und Programmsteuerung. Damit lassen sich festvorgegebene Experimentsteuerungen verwirklichen. Schwierigkeiten bereitet allerdings die Implementation der wichtigen Forderung nach einem direkten Zugriff zu allen CAMAC-Registern und Steuerparametern über den Fernschreiber. Dafür müßten auch besondere Bedienungsprogramme entwickelt werden.

2.3 Realzeitsprachen

Es gibt eine ganze Reihe von Vorschlägen für Realzeitsprachen /8,9/, von denen auch schon einige Probeimplementationen laufen /10,11/.

Neben einer Erweiterung der Datentypen, die auf eine bessere Anpassung an die Hardwarestrukturen der Prozeßsteuerungselemente und deren Ein-Ausgaben abzielen, sind in diesen Sprachen die Organisationen von parallelen Prozessen von größter Bedeutung. Der Programmierer erhält Sprachelemente zur zeitlichen Steuerung der Programme und zur Synchronisation verschiedener Programme. Die Realzeitsprachen müssen im Zusammenhang mit den Betriebssystemen implementiert werden, da sie einen wesentlichen Einfluß auf die Konstruktion des Betriebssystems ausüben.

Für den Benutzer bedeutet dies, daß er eine geschlossene Sprache für alle Probleme, die bei der Experimentsteuerung auftreten, erhält. Der besondere Vorteil liegt außerdem noch darin, daß diese Sprachen weitgehend computerunabhängig sein sollen ⁺⁾ , so daß die Standardisierung einer Realzeitsprache den Lernaufwand verringern könnte.

⁺⁾ aber noch abhängig von der Prozeßperipherie

Ähnlich wie REAL-TIME-FORTRAN benötigen die Realzeitsprachen Bedienungsprogramme für die Experimentautomatisierung, damit eine Steuerung vom Fernschreiber ermöglicht werden kann.

2.4 Interpretative Sprachen

Für Experimentsteuerungen sind bisher die Konzepte der interpretativen Sprachen FOCAL und BASIC am besten geeignet, um Programme zu erstellen. Deshalb wurden für sie für verschiedene Rechner Overlays zu den Compilern zur Bearbeitung von CAMAC entwickelt /12,13/.

Die interpretativen Sprachen besitzen die für die Experimentiertechnik wichtige Voraussetzung des sofortigen Zugriffs zu allen Parametern und ermöglichen außerdem die direkte Ausführung von Experimentsteuerbefehlen über den Fernschreiber. Durch die einfache, leicht erlernbare Struktur eignen sie sich besonders für die Anwenderprogrammierung.

Es gibt aber auch wichtige Gründe, die den Einsatz interpretativer Systeme begrenzen:

1. Die interpretativen Sprachen sind keine Realzeitsprachen. Aus diesem Grunde sind die Koordinationen zwischen dem Prozeß und den Programmen nur schwierig zu realisieren.
2. Die interpretativen Sprachen werden erst zur Laufzeit übersetzt, so daß sie dadurch für viele Fälle zu langsam sind.

Trotz dieser Nachteile können sie bei einer großen Zahl von Laborautomationsaufgaben eingesetzt werden, die keine allzu schwierigen Realzeitbedingungen haben, und bei denen die Ausführungszeiten der Programme kein wichtiges Kriterium für das System bedeuten.

2.5 Die CAMAC-Sprachen

Dem ESONE-Komitee, das die Spezifikationen von CAMAC herausgegeben hat, ist auch eine Software-Arbeitsgruppe beigeordnet, die sich mit den aus der CAMAC-Hardware entstehenden Softwarefragen befaßt. Von dieser Arbeitsgruppe sind zwei Sprachvorschläge für die Bearbeitung von CAMAC unterbreitet worden /14,15/.

Während der erste Vorschlag, die "CAMAC-Language" die Bedienung von CAMAC auf einem hohen Sprachniveau ermöglichen soll, ist die zweite Sprache, CAMAC-IML (CAMAC-Intermediate Language) eine assemblernahe Sprache. Für den Anwender hat die CAMAC-Language die wichtigere Bedeutung. Die CAMAC-IML könnte als Zwischenstufe zur Implementation der CAMAC-Language eingesetzt werden, wie es auch im Sprachvorschlag erwähnt wird.

Die CAMAC-Language beinhaltet Sprachdefinitionen für Datentypen, CAMAC-Adressen sowie CAMAC-Befehle. Ihre Konzeption sieht vor, daß sie in eine andere Sprache eingebettet werden soll und enthält deshalb auch keine arithmetische Sprachteile. Durch Vorschaltung eines Precompilers sollen die CAMAC-Sprachelemente in Anweisungen der Wirtssprache oder den Assemblercode übersetzt werden. Zur Synchronisation beider Compiler sind entsprechende Sprachelemente vorgesehen. Außerdem beinhaltet die CAMAC-Language Ausführungsanweisungen für eine Programmkoordination im Realzeitbetrieb.

Die CAMAC-Language setzt im allgemeinen als Wirtssprache eine Realzeitsprache voraus, zumindest ist aber ein Realzeitbetriebssystem erforderlich.

2.6 Zusammenfassende Beurteilung

Obwohl ein breites Spektrum auf dem Softwaregebiet der Realzeitverarbeitung bearbeitet wird, lassen sich die Anforderungen durch die Experimentiertechnik nur

unvollständig abdecken. Abgesehen von der Frage der Verfügbarkeit für den verwendeten Rechner sind ausnahmslos Eigenentwicklungen notwendig, um die Softwaresysteme den Bedürfnissen der Benutzer anzupassen.

Prinzipiell ist die Arbeitsweise, wie sie bei den interpretativen Systemen anzutreffen ist, nämlich die Erstellung und die Ausführung der Programme direkt vom Fernschreiber aus zu steuern, am besten für die Experimentiertechnik geeignet. Sie läßt dem Benutzer einen großen Spielraum in der Anwendung. Allerdings dürfen die Programme nicht erst zur Laufzeit übersetzt werden.

Dagegen reicht aber der Befehlsumfang der interpretativen Sprachen nicht aus, so daß Sprachelemente für die Bedienung von CAMAC, wie sie in der CAMAC-Sprache definiert sind, unerläßlich sind. Außerdem ist ein simultaner Programmbetrieb, wie er in den Realzeitsprachen vorgesehen ist, notwendig. Von der Bedienungssprache müssen daher entsprechende Sprachelemente zur Verfügung stehen.

3. Ein CAMAC-orientiertes Analysier- und Auswertesystem (CoAAS)

3.1 Zusammenstellung der Grundzüge des CoAAS

Da keines der vorhandenen Softwaresysteme den Anforderungen gerecht wird, wurde für den Einsatz der PDP 7 zur Experimentsteuerung und Datenerfassung ein CAMAC-orientiertes Analysier- und Auswertesystem entwickelt. Dieses System vereinigt die vorteilhaften Eigenschaften der verschiedenen Softwaresysteme zu einem einheitlichen Konzept, das auf die Anwendung zugeschnitten ist. Die Grundzüge des Systems lassen sich in den folgenden 11 Punkten zusammenfassen:

1. Der Benutzer hat über den Fernschreiber Zugriff zu allen Parametern. Anweisungen, insbesondere alle CAMAC-Befehle, können direkt ausgeführt werden.
2. Programme, genannt Tasks, können über den Fernschreiber oder über's DECTape eingegeben werden.
3. Tasks können vom Benutzer oder von CAMAC gestartet werden.
4. Die Tasks werden in den Binärcode übersetzt. Ihr Quellcode wird auf einem DECTape gespeichert, so daß er wieder geladen werden kann.
5. Die Sprachelemente zur Bedienung von CAMAC wurden der CAMAC-Sprache entlehnt.
6. Die Zahlendarstellungen und die Sprachelemente zur Programmierung sowie zur Ein- und Ausgabe der Daten wurden von FORTRAN übernommen.
7. Die CAMAC-Anweisungen werden in Macros übersetzt, die einen für das CAMAC-Interface entwickelten Devicedriver ansprechen.
8. Die Tasks laufen prioritätsgesteuert.
9. Das MONITOR-Betriebssystem wurde für die Realzeitverarbeitung modifiziert.

10. Die bisher bewährten Funktionen für die Analyse, Darstellung und Auswertung wurden übernommen und ins System integriert /27/.

11. Das Bildschirmprogramm läuft als Hintergrundprogramm.

3.2 Struktur des Systems

Die Struktur und die Handhabung des Systems sollen an ein paar exemplarischen Beispielen erläutert werden. Der volle Sprachumfang ist in der Sprachbeschreibung aufgeführt.

Im CoAAS können neben der Datenaufnahme und -darstellung die zwei folgenden Aufgaben bearbeitet werden:

1. Die Simulation einer Handbedienung von CAMAC.
2. Die Erstellung und der Lauf von Programmen zur Experimentsteuerung und Auswertung.

Für diese zwei Aufgaben sind zwei Betriebsmodi vorgesehen.

3.2.1 Simulation der Handbedienung

Im NIM-System werden die Parameter über Schalter und Potentiometer eingestellt, wobei ihre Stellung gleichzeitig eine Anzeige für den Experimentator ist. Die Parameter lassen sich auch während des Experiments jederzeit ändern.

Um diese Möglichkeiten auch im CAMAC-System zu erhalten, kann der Experimentator

1. einmalig einzustellende Parameter seiner Elektronik über den Fernschreiber direkt setzen,
2. im laufenden Experiment CAMAC-Register verändern und
3. Informationen über den Zustand seiner Elektronik ausgeben, ohne das Experiment zu unterbrechen.

Die Bedienung erfolgt durch direkte Ausführung von CAMAC-Anweisungen, die über den Fernschreiber eingegeben werden. Jedem CAMAC-Funktionscode F entspricht ein mnemotechnischer Ausdruck. Zusätzlich sind für die

vollständige Anweisung noch die CAMAC-Adresse sowie die Informationsquelle oder -senke bzw. die Information selbst notwendig.

Die Anweisung

WRITEC MOTOR,1000

Überträgt die Integerzahl 1000 in das CAMAC-Register mit der symbolischen Adresse MOTOR. Die symbolische Adresse muß vor dem Aufruf einer CAMAC-Anweisung eine Wertzuweisung erhalten, z.B.

MOTOR=6;1

Über diese Anweisung wird die symbolische Adresse MOTOR mit der aktuellen CAMAC-Adresse N=6 und A=1 geladen. (Weil z.Z. nur ein CAMAC-Crate angeschlossen ist, wird die Crateadresse C beim Aufruf automatisch gegeben. Sie darf deshalb auch nicht gesetzt werden.)

In der CAMAC-Anweisung kann die CAMAC-Adresse auch absolut gegeben werden, z.B.

WRITEC 6;1,1000

Zum Lesen eines CAMAC-Registers wird stets eine symbolische Adresse für die Aufnahme der Information benötigt. Die Anweisung

READC 7;2,ITIME

Überträgt den Inhalt des CAMAC-Registers der absoluten Adresse N=7 und A=2 in die Zelle ITIME als Integerzahl. Über eine Schreibanweisung, die sich auf eine vorher-eingegebene FORMAT-Anweisung bezieht, kann der Inhalt jedes symbolischen Parameters, der als Integerzahl in der Maschine dargestellt ist, auf dem Fernschreiber ausgegeben werden.

100 FORMAT (1H ,I6)

 WRITE (4,100)ITIME

Da die FORMAT-Anweisungen stets im Datenpuffer abgelegt werden, können sie für weitere Ein- und Ausgaben wieder benutzt werden.

Alle Anweisungen werden sofort ausgeführt, wenn das Betriebssystem im direkten Ausführungsmodus ist.

Es bestehen für den Experimentator zwei Möglichkeiten, in ein Experiment einzugreifen,

1. durch direkte Ausführung von CAMAC-Befehlen,
2. durch Veränderung von symbolischen Integer-variablen, die als Parameter in Programmen zur Einstellung von CAMAC-Registern benutzt werden.

Die Arbeitsweise im direkten Ausführungsmodus ähnelt der in einer interpretativen Sprache. Im Gegensatz dazu wird allerdings jede Anweisung erst in einen Binärcode übersetzt, der dann ausgeführt wird.

3.2.2 Die Taskerstellung

Alle Benutzerprogramme werden als Task eingegeben, die einzeln gestartet werden können. Zur Eingabe der Tasks wird das Betriebssystem in den Taskerstellungsmodus umgeschaltet. Alle im Taskerstellungsmodus eingegebenen Anweisungen werden übersetzt und in einem Programmpuffer abgelegt. Gleichzeitig mit der logischen Beendigung einer Task wird der Taskerstellungsmodus verlassen und das Betriebssystem kehrt in den direkten Ausführungsmodus zurück.

3.2.2.1 Die Open-Anweisung

Alle Anweisungen, die im Taskerstellungsmodus eingegeben werden, werden in den Binärcode übersetzt, während sie gleichzeitig, wenn sie syntaktisch richtig sind, auf einem DECTape (Unit 2) abgespeichert werden. Deshalb muß vor der Erstellung einer Task ein File auf dem DECTape eröffnet werden. Dies geschieht durch die Anweisung

OPEN name

name repräsentiert den Filenamen, der den Vorschriften des MONITOR-Betriebssystems entsprechen muß. Als

Extension wird SRC gesetzt. Tasks, die auf einem DECTape abgespeichert sind, können wieder geladen werden, nachdem der entsprechende File mit der OPEN-Anweisung eröffnet wurde.

3.2.2.2 Die TASK-Anweisung

Durch die Anweisung

TASK n

,wobei n eine Integerzahl zwischen 1 und 1000 sein kann, wird

1. eine Task definiert, die den Namen n erhält und
2. der Übergang des Betriebssystems vom direkten Ausführungsmodus in den Taskerstellungsmodus vollzogen.

Wenn die TASK n bereits als Quellfassung auf dem durch die OPEN-Anweisung eröffneten File steht, kann sie vom DECTape geladen werden. Ansonsten werden alle Anweisungen, die nach dem Übergang in den Taskerstellungsmodus durch die TASK-Anweisung gegeben werden, in den Binärcode übersetzt und in einem Programmpuffer gespeichert, während der Quelltext auf dem DECTape im eröffneten File abgelegt wird.

Die Tasknummer legt außerdem noch die Priorität fest, wobei die kleinste Zahl die höchste Priorität einer Task festlegt.

Im Taskerstellungsmodus legt die Reihenfolge der Eingabe der Anweisungen auch die Reihenfolge der Abarbeitung bei der Ausführung fest. Diese Reihenfolge läßt sich nur durch Sprungbefehle verändern.

3.2.2.3 Die END- und RETURN-Anweisungen

Der Taskerstellungsmodus wird nach der logischen Beendigung eines Programms durch die Anweisung

END

verlassen. Damit wird auch gleichzeitig das bis dahin erstellte Programm als Task abgeschlossen, so daß es

zur Ausführungszeit auch an dieser Stelle ordnungsgemäß beendet werden kann.

Eine Beendigung einer Task zur Laufzeit erfolgt durch die Anweisung

RETURN

Damit wird allerdings der Taskerstellungsmodus nicht verlassen.

3.2.2.4 Starten einer Task

Tasks können auf zwei Arten gestartet werden.

1. Start durch den Benutzer

Durch den Befehl

S n

der auf den Fernschreiber gegeben wird, kann durch den Benutzer die Task mit der Nummer n gestartet werden.

2. Start von CAMAC

Im Devicedriver ist eine Liste von Sprungadressen vorgesehen, in der jeder CAMAC-Platz eines Crates einen Platz entsprechend der Reihenfolge der Modulplätze 1 bis 22 belegt. Durch die Anweisung

ON DEMAND n DO m

wird in dieser Sprungliste auf Platz n die Sprungadresse auf die Task m eingetragen.

Dadurch wird nach der Erkennung eines LAM's vom Modulplatz n die Task mit der Nummer m sofort bearbeitet.

An dieser Stelle muß die sofortige Bearbeitung betont werden, d.h. diese Bearbeitung erfolgt nicht unter der Kontrolle der Tasksteuerung.

Jede Task, die durch ein LAM angestoßen wird, muß deshalb einen Kopf erhalten, indem sie ohne Tasksteuerung laufen kann.

Dieser Kopf ist notwendig, damit die Ursache eines LAM's oder zumindest das LAM selber vom Benutzer über CAMAC-Anweisungen abgeschaltet werden kann. Erst nach dieser Operation kann die Task der Tasksteuerung übergeben werden. Dies erfolgt durch die Anweisung

TAKE

Vor der TAKE-Anweisung dürfen keine arithmetischen Ausdrücke oder FORTRAN-Ein-Ausgaben durchgeführt werden. Lediglich CAMAC-Anweisungen, Integerzuweisungen einfacher Variabler und Sprünge dürfen vor einer Übergabe der Task an die Tasksteuerung gegeben werden.

Durch die TAKE-Anweisung wird gleichzeitig der Interrupt von CAMAC abgeschlossen, so daß ein nächster Interrupt bearbeitet werden kann, allerdings nicht von dieser Task. Sie muß erst vollständig abgearbeitet sein, ehe sie wieder angestoßen werden kann.

3.2.2.5 Beispiel einer Task

Das folgende einfache Beispiel zeigt die allgemeine Struktur einer Task:

	TASK 3	/3=Nummer der Task und /Priorität
	READC 1;0,K	/lies das CAMAC-Register 1;0 /nach K und lösche LAM
	TAKE	/Übernahme in die Tasksteuerung
	IF K 1,1,2	/Wenn K=0, dann drucke K sonst
1	CLEAR 1;2	/Initiiere nächste Eingabe
	RETURN	/beende Task (dynamischer /Abschluß)
100	FORMAT (1H,6I)	/Formatangabe vor der /Schreibanweisung
2	WRITE(4,100)K	/Schreibe K auf Fernschreiber /aus
	GOTO 1	/Sprung
	END	/logischer Abschluß der Task /hier kein dynamischer Abschluß

4. Sprachzusammenfassung

4.1 Zahlen- und Adreßdarstellungen

Wie in FORTRAN gibt es im CoAAS Integer- und Realzahlen. Während die Realzahlen unverändert übernommen wurden, sind die Integerzahlen durch Oktalzahlen und CAMAC-Adressen erweitert worden. Für jede Zahl wird im sogenannten Variablenpuffer Speicherplatz reserviert. Sie sind globale Größen, d.h. sie werden für alle Tasks nur einmal abgespeichert und alle Tasks haben darauf Zugriff.

4.1.1 Konstanten

Konstanten sind fest vereinbarte Größen in Anweisungen. Es gibt vier Arten, die auch im Variablenpuffer aber unter dem Kennzeichen einer Konstanten abgespeichert werden.

4.1.1.1 Ganzzahlige Konstanten

4.1.1.1.1 Integerkonstanten

Die Integerkonstanten entsprechen der Definition im FORTRAN.

4.1.1.1.2 Oktalkonstanten

Oktalkonstanten sind zur Eingabe von Bitmustern gedacht. Sie werden durch ein vorangestelltes Dollarzeichen gekennzeichnet. Danach können 1 bis 6 Oktalziffern folgen, die rechtsbündig abgelegt werden.

Beispiel: Die Oktalkonstante $\$372$

liefert das Bitmuster 000 000 000 011 111 010

und ist identisch mit der Integerkonstanten 250.

Oktalkonstanten sind wie Integerzahlen zu behandeln.

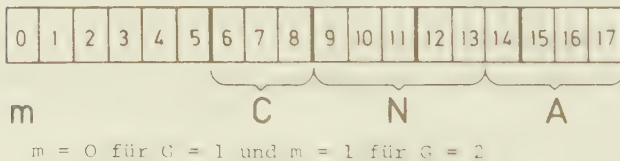
4.1.1.1.3 CAMAC-Adreßkonstanten

Die CAMAC-Adreßkonstanten haben die allgemeine Form
<CAMAC-Adreßkonstante> ::=c;n;aGm

Hierbei bedeuten c die Cratenummer 1 bis 7, n die Stationsnummer 1 bis 22 und a die Subadresse 1 bis 16.

Im CAMAC-System sind zwei Registergruppen 1 und 2 vorgesehen, die mit verschiedenen Funktionen behandelt werden. Genau wie in der CAMAC-Sprache wird die Gruppeneinteilung aus den Funktionen herausgenommen und der Adresse zugefügt, da dies der logischen Bedeutung der Gruppe entspricht. Gm bezeichnet die Gruppe 1 oder 2. Da die meisten Module mit der Gruppe 1 arbeiten, kann in diesem Fall G1 fortgelassen werden. G1 wird dann automatisch zugefügt. (G2 muß aber stets angegeben werden.) Da z.Z. auch nur ein Crate angeschlossen ist, wird auch die Crateadresse (6) automatisch erzeugt und es darf deshalb kein C gegeben werden.

Die CAMAC-Adreßkonstanten werden wie Integerzahlen behandelt und es ist deshalb die interne Darstellung wichtig. Insbesondere bei der Ausnutzung von Laufanweisungen zur CAMAC-Adressierung ist diese Information nötig.



4.1.1.2 Realzahlkonstanten

Die Realzahlkonstanten entsprechen der Definition von FORTRAN. Eine Zahl belegt 2 Speicherzellen im Variablenpuffer (2-Wortdarstellung).

4.1.2 Variablen

Im CoAAS werden in den Anweisungen drei Arten von Variablentypen benutzt:

1. Realzahlvariablen
2. Intervariablen
3. CAMAC-Adreßvariablen

Während die Realzahl- und Intervariablen als einfache Variablen und Feldvariablen definiert werden können, lassen sich CAMAC-Adreßvariablen nur als einfache Variablen benutzen.

4.1.2.1 Einfache Variablen

4.1.2.1.1 Realzahl- und Integervariablen

Gegenüber FORTRAN besteht kein Unterschied in der Definition von Realzahl- und Integervariablen. Realzahlvariablen werden in der 2-Wort-Darstellung abgespeichert. Alle einfachen Variablen werden implizit beim ersten Auftreten ihres Namens in einer Anweisung definiert und es wird gleichzeitig Speicherplatz im Variablenpuffer reserviert.

4.1.2.1.2 CAMAC-Adreßvariablen

Als CAMAC-Adreßvariablen sind einfache Integervariablen zulässig, die auch genau wie diese behandelt und definiert sind. Von den CAMAC-Adreßvariablen wird lediglich verlangt, daß sie als Inhalt eine CAMAC-Adresse enthalten. Durch die Verwendung des Formats einer Integervariablen auch als Adreßvariable können CAMAC-Register, die in Gruppen angeordnet sind, leicht über Laufanweisungen mit entsprechender Verwendung der Laufvariablen adressiert werden. CAMAC-Adreßvariablen werden implizit definiert. Zur internen Darstellung siehe CAMAC-Adreßkonstanten.

4.1.2.2 Datenfelder

Wie in FORTRAN gibt es Datenfelder für Integer- und Realzahlvariablen, die explizit durch eine oder mehrere DIMENSION-Anweisungen deklariert werden müssen. Es sind maximal dreidimensionale Felder erlaubt. Im Gegensatz zu einfachen Variablen können Integerfeldelemente nicht zur Adressierung von CAMAC benutzt werden.

4.1.2.3 Datenbereiche der Spektren

Einen Zusatz zur Felddefinition bilden die Datenbereiche für Spektren. Sie werden implizit durch die Festlegung der Konfiguration über den Befehl KNF definiert (vergl. 4.12.5). Entsprechend der Anzahl der Bereiche von 1 bis maximal 8 sind Felder mit dem Namen IB1 bis IB8 definiert, deren

Längen durch die Spektrumslängen der Bereiche bestimmt sind. Die Felder IB1, IB2, ... werden wie Integerfelder behandelt. Dadurch können über CAMAC gelesene Daten sofort in Spektrumsfelder gespeichert und sichtbar gemacht werden.

4.2 Arithmetische Ausdrücke

In den arithmetischen Ausdrücken ist die Verwendung von Konstanten, einfachen und dimensionierten Variablen gleichen Typs zulässig. Nach dem Zuweisungsoperator kann eine Typänderung erfolgen. Die arithmetischen Ausdrücke können nach den Vorschriften der Algebra beliebig geklammert werden, allerdings dürfen keine zwei Operatoren aufeinander folgen. Als Operatoren sind zulässig:

<Operatoren> ::= + - * /

Die Abarbeitung der Ausdrücke erfolgt von links nach rechts. Zur Indexberechnung sind nur maximale Ausdrücke der Form

$n * m + 1$

zulässig.

4.3 Laufanweisung

Die Laufanweisung hat die gleiche Form wie in FORTRAN

<Laufanweisung> ::= DO n i=j,k,l

Ist das Inkrement l gleich 1 so kann es weggelassen werden.

4.4 Sprunganweisungen

Im CoAAS gibt es zwei Sprunganweisungen.

4.4.1 Unbedingte Sprunganweisung

Der unbedingte Sprung erfolgt über die Anweisung

GOTO n

wobei n die Anweisungsnummer der Zielanweisung ist. (Es kann nicht auf FORMAT-Anweisungen gesprungen werden.)

4.4.2 Bedingte Sprunganweisung

Die bedingte Sprunganweisung ist eine Vereinfachung der arithmetischen Sprunganweisung von FORTRAN. Sie hat die Form:

IF v n_1, n_2, n_3

V ist eine Integer- oder Realvariable und n_1 , n_2 und n_3 sind Anweisungsnummern, wobei n_1 angesprungen wird, wenn $v < 0$, n_2 angesprungen wird, wenn $v = 0$, und n_3 angesprungen wird, wenn $v > 0$ ist.

4.5 Standard Ein-Ausgabe-Anweisungen

Zur formatierten Ein-Ausgabe auf den Standardgeräten werden die READ- und WRITE-Anweisungen wie in FORTRAN unter Hinzuziehung eines FORMAT's verwendet. Zu beachten ist dabei, daß die FORMAT-Anweisung stets gegeben sein muß, bevor eine Ein-Ausgabe-Anweisung eingegeben wird. Diese Vorschrift ist notwendig, weil Ein-Ausgabe-Anweisungen auch im direkten Ausführungsmodus gegeben werden können. Die logische Gerätenummer wird nach den Vorschriften des MONITOR-Systems benutzt. In den Ein-Ausgabe-Anweisungen sind keine Schleifenanweisungen zugelassen.

4.6 Die FORMAT-Anweisung

Eine FORMAT-Anweisung muß stets eine Anweisungsnummer tragen. Der Aufbau der Anweisung, sowie die möglichen Formatschlüssel entsprechen der FORTRAN-Beschreibung in /14,15/.

4.7 Die Anweisungsnummern

Jede Anweisung kann mit einer Anweisungsnummer versehen werden, deren Wertebereich zwischen 1 und 100000 liegen sollte. Im Unterschied zu FORTRAN werden die Anweisungsnummern für FORMAT-Anweisungen intern besonders gekennzeichnet, da sie im Datenpuffer abgelegt werden, so daß jede Nummer sowohl als Anweisungsnummer vor

FORMAT-Anweisungen als auch vor anderen Anweisungen benutzt werden kann. Syntaktisch ist als Trennzeichen nach der Anweisungsnummer ein Zwischenraumzeichen oder ein CNTL TAB gefordert. CNTL TAB kann vor jeder Anweisung stehen.

4.8 Die Leeraanweisung

Wie in FORTRAN lautet die Leeraanweisung
CONTINUE

4.9 Kommentaranweisung

Zur besseren Dokumentation der Quellprogramme können in jedes Programm Kommentare eingefügt werden. Sie müssen mit einem C und einem nachfolgenden Zwischenraum beginnen
C beliebiger Kommentar.

4.10 CAMAC-Anweisungen

4.10.1 Allgemeine Beschreibung

Über CAMAC-Anweisungen werden die Steuerung von CAMAC und der Datenverkehr mit dem CAMAC-System durchgeführt. Ein Teil der Anweisungen stützt sich direkt auf den Hardwarebefehlsumfang der genormten CAMAC-Funktionen, während ein zweiter Teil Befehle an den computer-spezifischen CAMAC-Kontroller bzw. an den dazugehörigen Devicedriver darstellen. Da diese Anweisungen in der Bedienungssprache nicht unterschieden werden, wird bei der Definition der entsprechenden Befehle darauf hingewiesen.

Während bei den Standardein- und -ausgabegeräten eine Formatierung nötig ist, wurde bei den CAMAC-Anweisungen keine Formatierung vorgenommen. Vorläufig wird CAMAC überwiegend zur Experimentsteuerung eingesetzt und dabei kommt man mit einem Bitstringübertrag im allgemeinen aus. Eine Ausnahme bildet allerdings die Behandlung von BCD-Registern, für die besondere Sprachelemente vorgesehen wurden.

Die CAMAC-Anweisung hat die allgemeine Form:

<CAMAC-Anweisung> ::= Befehl ICNA, IADR

Der Befehl besteht aus einem 4 bis 6 Buchstaben umfassenden Befehlswort, ICNA ist die direkte oder symbolische CAMAC-Adresse und IADR ist eine symbolische Kernspeicheradresse. ICNA und IADR sind entsprechend der Definition der Datentypen Integerzahlen.

Da für den CAMAC-Kontroller an der PDP 7 keine Datenkanäle zur Verfügung stehen, wurden auch keine allgemeinen Anweisungen für den Blocktransfer vorgesehen. Auch eine Simulation ist wenig sinnvoll, da praktisch der wesentliche Zeitbedarf bei der Abhandlung des Interrupts benötigt wird. Deshalb sollte der Benutzer selbst die Organisation vornehmen, wenn er sie benötigt. Eine Ausnahme bilden allerdings die ADC's.

4.10.2 Unterteilung in Gruppen

Die CAMAC-Anweisungen unterteilen sich entsprechend der benötigten Zusatzinformation bei der Ausführung in 3 Gruppen. Während die erste Gruppe allgemeine Steueranweisungen enthält und deshalb auch keine zusätzlichen Adressen benötigt, werden in der 2. Gruppe eine CAMAC-Adresse und in der dritten Gruppe sowohl die CAMAC- als auch die Speicheradresse benötigt.

4.10.2.1 Anweisungen der ersten Gruppe

Die Anweisungen der ersten Gruppe sind Steuerbefehle an den CAMAC-Kontroller. Sie bestehen nur aus einem Befehlsteil. Folgende Anweisungen sind zulässig:

1.	INIT	(Z)
2.	INHIBT	(I)
3.	CLRINH	(I not)
4.	ZEROAL	(C)
5.	INITC	
6.	CLOSEC	

Die Befehle 1 bis 4 entsprechen den gemeinsamen Steuerungssignalen Initialisieren (Z), Sperren (I), Sperre löschen (I not) und Löschen (C).

Die Befehle 5 und 6 sind keine standardisierten CAMAC-Funktionen. Sie sind Anweisungen an den CAMAC-Driver und entsprechen sinngemäß der Initialisierung eines Device-Handlers, wie er in der MACRO-Sprache erforderlich ist. INITC bindet den Devicedriver in die Skipkette ein und schaltet gleichzeitig den Interrupt für das CAMAC-Interface ein. CLOSEC schaltet den Interrupt des CAMAC-Interfaces aus, so daß danach kein LAM-Signal einen Interrupt in der PDP 7 erzeugen kann /23/.

4.10.2.2 Anweisungen der zweiten Gruppe

Diese Gruppe von CAMAC-Befehlen enthält Steuerbefehle an die Module. Aus diesem Grunde muß zum Befehl noch die Moduladresse angegeben werden.

Folgende Anweisungen sind definiert:

	Befehlswort	CAMAC-Funktion F
1.	CLEAR	F(9) u. F(11)
2.	CLEARL	F(10)
3.	ENABLE	F(26)
4.	DISABL	F(24)
5.	TSTSTA	F(27)
6.	EXECUT	F(25)
7.	TSTLAM	F(8)

Alle Befehle entsprechen den definierten CAMAC-Funktionen.

4.10.2.3 Anweisungen der dritten Gruppe

Die Anweisungen der dritten Gruppe bewältigen den Datenverkehr zwischen dem Rechner und CAMAC. Deshalb müssen zwei Adressen, d.h. CAMAC- und Kernspeicheradresse angegeben werden. Zu dieser Gruppe gehören folgende Befehle:

	Befehlswort	CAMAC-Funktion F
1.	READC	F(0) + F(1)
2.	READCL	F(2)
3.	READCM	F(3)
4.	WRITEC	F(16) + F(17)
5.	WRITES	F(18) + F(19)
6.	RDBCD	
7.	WTBCD	

einen 16-Bit-Übertrag ausgelegt, und ein voller 24-Bit-Übertrag wird durch zusätzliche Anweisungen unmittelbar vor bzw. nach dem CAMAC-Befehl bearbeitet.

Außerdem kann jeder Modul beim Ansprechen durch einen CAMAC-Befehl eine Ein-Bit-Information Q übertragen. Dieser Zustand kann ebenfalls unmittelbar nach einer CAMAC-Anweisung abgefragt werden.

4.10.4.1 Schreiben der obersten 6 Bit

Zum Schreiben des vollen 24-Bit-Wortes werden mit der Anweisung

SET HIBIT=k

die untersten 6 Bit der Integerkonstanten oder -variablen k in die 6 höchstwertigen Bits gesetzt und zusammen mit dem in der unmittelbar folgenden CAMAC-Schreibanweisung Datenwort übertragen.

4.10.4.2 Lesen der obersten 6 Bit

Nach jeder CAMAC-Leseanweisung können mit der Anweisung

SET k=HIBIT

die obersten 6 Bit der gelesenen Daten in die untersten Bits der Integervariablen k übertragen werden.

4.10.4.3 Die Q-Abfrage

Unmittelbar nach jeder CAMAC-Anweisung kann durch die beiden Anweisungen

IFQ GOTO n

IFNQ GOTO m

der Zustand von Q durch einen Sprung analysiert werden.

Der Sprung auf die Anweisungsnummer n wird bei IFQ ausgeführt, wenn Q=1 gesetzt ist, während bei IFNQ der Sprung auf m ausgeführt wird, wenn Q=0 gesetzt ist.

Die Anweisungen SET k=HIBIT und IFQ GOTO n bzw. IFNQ GOTO m können, wenn sie beide nach einer Leseanweisung benötigt werden, in beliebiger Reihenfolge gegeben werden.

4.11 Übergeordnete Anweisungen

Zu den übergeordneten Anweisungen gehören die Compileranweisungen sowie Anweisungen, die die statische und dynamische Programmstruktur festlegen.

4.11.1 Fileeröffnungsanweisung

Der Quellcode aller Tasks wird auf einen DECTape mit der UNIT 1 abgespeichert. Mit der Anweisung

OPEN name

wird ein File mit dem Filenamen name geöffnet bzw. neu eröffnet. Der Filename darf aus maximal 6 Buchstaben und Ziffern bestehen, wobei als erstes Zeichen stets ein Buchstabe stehen muß. Der File erhält die Extension SRC.

4.11.2 Löschen eines Files

Mit der Anweisung

DELETE FILE name

wird der File mit dem Filenamen name gelöscht, der auf dem DECTape der UNIT 1 liegt.

4.11.3 Löschen des Directorys

Mit der Anweisung

DELETE ALL

wird ein neues Directory auf ein DECTape der UNIT 1 geschrieben. Damit werden gleichzeitig alle File's gelöscht.

4.11.4 Einrichtung einer Task

Durch die Anweisung

TASK n

wird ein Task mit der Tasknummer n eröffnet. Die Nummer n darf zwischen 1 und 1000 liegen und gibt gleichzeitig die Priorität der Task an, wobei die kleinste Zahl die höchste

Priorität ergibt. In Abhängigkeit davon, ob die Task in der durch die OPEN-Anweisung eröffneten File vorhanden ist oder nicht, geht das Betriebssystem in zwei Zustände über.

1. Taskerstellungsmodus

Dann ist ein File mit der Filenummer n noch nicht auf dem eröffneten File vorhanden, und es kann eine neue Task erstellt werden. Alle danach eingegebenen Anweisungen werden in den Binärcode übersetzt, bis eine END-Anweisung erfolgt. Der Quelltext der Task wird gleichzeitig auf dem eröffneten File abgespeichert.

2. LOAD-Modus

Dann ist die Task mit der Nummer n bereits als Quellversion auf dem eröffneten File abgespeichert, und es wird LOAD auf dem Fernschreiber ausgedruckt.

Der Benutzer kann in diesem Fall mit der Anweisung

LOAD

die aufgerufene Task laden, d.h. sie wird vom Band gelesen, übersetzt und im Programmpuffer abgespeichert. Damit ist es möglich, eine im Taskerstellungsmodus erstellte Task, später wieder zu laden.

Mit der Anweisung

PRINT

wird die aufgerufene Task nur ausgedruckt, aber nicht übersetzt.

Nach der nächsten END-Anweisung geht das Betriebssystem wieder in den direkten Ausführungsmodus über.

Sollte eine Task n bereits bestehen, wenn eine neue Anweisung TASK n gegeben ist, so wird angenommen, daß der Quelltext ausgedruckt werden soll. Ist der Quelltext auf dem eröffneten File vorhanden, so wird EDIT ausgedruckt.

Durch die Anweisung

PRINT

wird der Text der Task ausgedruckt.

Ist die TASK n nicht auf dem eröffneten File vorhanden, so wird eine Fehlermeldung gegeben und das Betriebssystem geht in den direkten Ausführungsmodus über.

4.11.5 Beendigung einer Task

Die Anweisung

END

setzt den Code für die dynamische Beendigung einer Task ab. Gleichzeitig wird der Taskerstellungsmodus abgeschlossen und das Betriebssystem kehrt in den direkten Ausführungsmodus zurück. Außerdem wird der File mit der neuen Task ordnungsgemäß abgeschlossen. Anschließend wird der File wieder geöffnet.

Die Anweisung

RETURN

setzt den Code für eine dynamische Beendigung einer Task ab. Dadurch wird aber der Taskerstellungsmodus nicht verlassen.

4.11.6 Die TAKE-Anweisung

Eine Task kann direkt durch ein LAM von CAMAC angestoßen werden. Die erste Aufgabe einer Task muß dann sein, die LAM-Quelle zurückzusetzen. Erst danach darf die Task der Taskverwaltung übergeben werden.

Die Anweisung

TAKE

veranlaßt die Übernahme dieser Task in die Taskverwaltung. Gleichzeitig wird damit auch der CAMAC-Driver für die Entgegennahme eines nächsten Interrupt freigegeben.

Wichtig! Zwischen der TASK- und der TAKE-Anweisung dürfen nur CAMAC-Anweisungen, undimensionierte Integerzuweisungen und Sprunganweisungen gegeben werden.

4.11.7 Der Anschluß einer Task an CAMAC

Im Devicedriver ist eine Liste von Sprungadressen vorgesehen, in der jeder CAMAC-Platz des Crates einen Platz in der Liste entsprechend der Reihenfolge der Modulplätze 1 bis 22 belegt. Durch die Anweisung

ON DEMAND n DO m

wird in dieser Sprungliste auf Platz n die Sprungadresse auf die Task mit der Nummer m eingetragen, so daß nach der Erkennung eines LAM's vom Platz n der Devicedriver die Bearbeitung der Task m übergibt.

Solange eine Task nur durch ein LAM gestartet ist, läuft sie auf höchster Priorität ohne Benutzung der Tasksteuerung. Erst durch die Anweisung TAKE wird die Task der Tasksteuerung übergeben, so daß sie danach prioritätsgesteuert bearbeitet wird.

4.11.8 Der Start einer Task

Eine Task kann mit der Anweisung

S n

wobei die Nummer der Task ist, gestartet werden. Der Start kann vom Fernschreiber oder durch eine andere Task erfolgen. Wenn die aufgerufene Task bereits läuft, wird der Befehl ignoriert.

4.12 Zusammenfassung der Analysier- und Auswertebefehle

Aus dem AAS /27/ wurden ins CoAAS die Befehle für die Analyse und die Auswertung von Spektren übernommen. Sie können jederzeit zusammen mit anderen Anweisungen in einer Task benutzt werden. Sie werden dann genau wie diese zur Laufzeit ausgeführt. Dadurch ist es möglich, auch automatische Auswerteprogramme zu erstellen.

Um diese Möglichkeit intensiv nutzen zu können, ist es zulässig, neben konstanten Argumenten auch symbolische Argumente zu geben, die vom Datentyp Integervariable sein müssen. Im folgenden soll ein Überblick über den Befehlsvorrat gegeben werden, für den prinzipiell die Beschreibungen der Bedienungsanleitung des AAS noch gültig sind (vgl. Programm SNG und SMB).

Als wichtige Änderung neben der Zulassung von Variablen als Argumente ist zu beachten, daß der Befehl und das nachfolgende Argument stets durch einen Zwischenraum getrennt sein müssen.

4.12.1 Bildschirmbefehle

Außer den Bildschirmbefehlen des SNG wurden auch die zusätzlichen Befehle für die Bearbeitung von Teilspektren aus dem SMB übernommen. Die Tabelle 3 gibt einen Überblick über die Bildschirmbefehle.

Befehl	Kurzbeschreibung
D n	Stelle den Bereich n dar
LG	Stelle Spektrum logarithmisch dar
LN m	Stelle Spektrum linear dar
ME m,n	Markeneingabe
MA	Markenausgabe
E	Expansion des Spektrums
SLG n	Länge n eines Teilspektrums
A n	Expandiere n-tes Teilspektrum

m und n können Integerkonstanten oder -variablen sein.

Tabelle 3: Bildschirmbefehle

4.12.2 Analysierbefehle

Neben den Analysierbefehlen über CAMAC kann eine Analyse auch über die Nuclear Data ADC's bearbeitet werden. Die folgende Tabelle 4 gibt einen Überblick über die Befehle.

Befehl	Kurzbeschreibung
X m	Starte Analyse auf der X-Seite
Y n	Starte Analyse auf der Y-Seite
X m, n	Starte Analyse auf X- und Y-Seite
NX	Stoppe Analyse auf der X-Seite
NY	Stoppe Analyse auf der Y-Seite
N	Stoppe Analyse auf X- und Y-Seite

n und m können Integerkonstanten oder -variablen sein.

Tabelle 4: Analysierbefehle

4.12.3 DECTapebefehle

Die DECTapebefehle wurden unverändert vom SNG übernommen.

Befehl	Kurzbeschreibung
INI 7	Initialisiere Datenband auf Unit 7
WD m	Schreibe Bereich m aufs Datenband
RD m, n	Lies Spektrum m vom Daten in den Bereich n

n und m können Integerkonstanten oder -variablen sein.

Tabelle 5: DECTapebefehle

4.12.4 Arbeitsbefehle

Als Arbeitsbefehle wurden Befehle aus dem SNG und SMB zusammengestellt.

Befehl	Kurzbeschreibung
Z n	Lösche Bereich n
I	Integriere zwischen den Lichtmarken
MS n	Speichere Marken nach Markenspeicher n
MI n	Marken holen aus Markenspeicher n
M	letzte Marken speichern, nächste Marken holen
MZ	Drucke Stellung des Markenzeigers
ERA	Lösche Markenliste
PL	Plotte das Spektrum auf dem Bildschirm

n kann eine Integerkonstante oder -variable sein.

Tabelle 6: Arbeitsbefehle

4.12.5 Konfigurationsbefehle

Als Datenbereiche stehen im CoAAS 8 K-Kernspeicher zur Verfügung, die in maximal 8 Datenbereiche eingeteilt werden können. Auch die dimensionierten Variablen und nur diese Variablen werden in diesen 8 K abgelegt, so daß sich leicht ein Überblick über die Kernspeicherbelegung für Daten gewinnen läßt.

Befehl	Kurzbeschreibung
BER	Drucke Bereichseinteilung
KNF $n_1, \dots, n_8$	neue Bereichskonfiguration

n_1 bis maximal n_8 sind die Länge der Datenbereiche und müssen duale Integerkonstanten sein.

Tabelle 7: Konfigurationsbefehle

Der Befehl KNF kann nur bis zur Eingabe der ersten DIMENSIONS-Anweisung gegeben werden, weil sonst Daten überschrieben werden könnten.

5. Struktur des Übersetzers

Das Softwaresystem CoAAS enthält zwei wesentliche Teile:

1. Den Übersetzer und
2. das Betriebssystem mit den nötigen Betriebsprogrammen.

In diesem Kapitel wird die Struktur des Übersetzers beschrieben.

Damit sowohl eine vollständige Task als auch Einzelanweisungen übersetzt werden können, wurde der Compiler als inkrementaler Übersetzer aufgebaut. Dadurch können auch Compileranweisungen übersetzungstechnisch auf die gleiche Art wie Ausführungsanweisungen behandelt werden.

Die Übersetzung wird anhand von Schlüsselworten in den Anweisungen durchgeführt. Nach ihrer Erkennung wird jede Anweisung der entsprechenden Semantikroutine zugeführt, die die Codeabsetzung vornimmt. Eine Sonderstellung nehmen die arithmetischen Anweisungen ein, weil sie keine Schlüsselwörter enthalten.

Der Compiler ist in Funktionsblöcke unterteilt, die zum Teil auch als selbständige Programme arbeiten könnten. In der Reihenfolge des Durchlaufs und der Benutzung einer Anweisung setzt er sich aus den folgenden sieben Teilen zusammen:

1. Das Eingabeprogramm mit Schlüsselworterkennung
2. Die Schlüsselworttabelle
3. Das Syntaxanalysierprogramm
4. Die Semantikroutinen
5. Die Listebearbeitungsroutinen und die Codeabsetzungsroutinen mit Zwischenpuffer
6. Die Übersetzungsroutine für arithmetische Ausdrücke
7. Die Abschlußsteuerung.

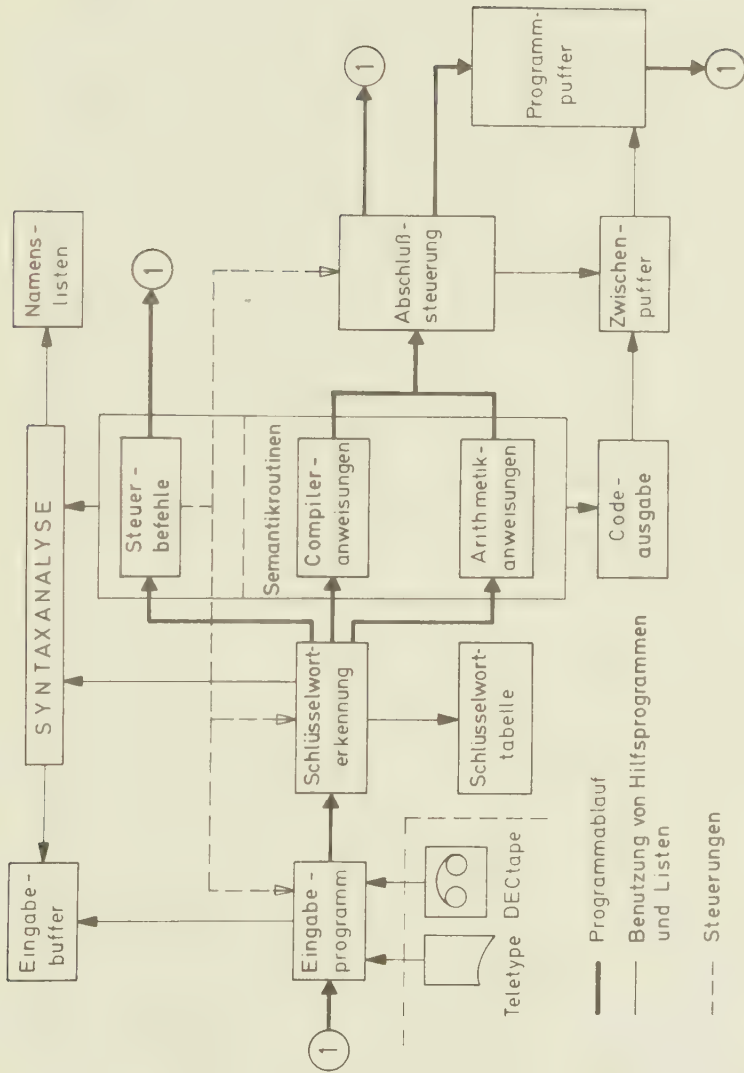


Abb. 2: Blockdiagramm des Übersetzers

Die Abb. 2 zeigt die Struktur des Compilers, der innerhalb des Betriebssystems als eine Task abläuft. Eine genauere Beschreibung befindet sich im Anhang E.

Eine Anweisung, die über das Eingabeprogramm vom DECTape oder Fernschreiber eingegeben werden kann, wird im Eingabepuffer abgelegt. Über die Schlüsselworterkennungs-routine, die die Hilfsroutine der Syntaxanalyse mit lexikalischer Analyse benutzt, werden die Anweisungen den Semantikroutinen bzw. der Routine zur Übersetzung arithmetischer Anweisungen zugeführt. Auf gleicher Ebene werden auch Steuerbefehle für den Compiler behandelt. Die Verwendung von unterschiedlichen Schlüsselworttabellen, die den Sprachumfang festlegen, ermöglicht die automatische Erzeugung von Varianten des Compilers.

Die Übersetzungsroutinen haben auch Zugriff zum Syntaxanalyseprogramm, das auch die Absetzung von Namen in Namenslisten sowie die Adreßzuweisung vornimmt.

Der generierte Code wird bis zur fehlerfreien Übersetzung einer Anweisung über die Codeausgabe in einem Zwischenpuffer gesammelt.

Während Steuerbefehle für den Compiler sofort ausgeführt werden und danach das Eingabeprogramm fortgesetzt wird, laufen alle anderen Anweisungen über die Abschlußsteuerung, die den Zwischenpuffer in den Programmpuffer überträgt. Je nach Zustand des Compilers wird danach der übersetzte Code ausgeführt oder das Eingabeprogramm sofort angesteuert. Die Abschlußsteuerung nimmt auch die Fortschaltung der Programmbelegung im Programmpuffer vor, wenn der Compiler im Programmeingabemodus ist. Im direkten Ausführungsmodus dagegen bleibt der Zeiger für den freien Kernspeicherplatz unverändert, so daß der übersetzte Code der folgenden Anweisung an die gleiche Stelle gelegt wird.

6. Das Betriebssystem

Das Betriebssystem für das CoAAS ist auf dem Advanced-Software-System (ADSS) aufgebaut, das als Grundbetriebssystem für die PDP 9 von der Digital-Equipment-Corporation entwickelt und von dem auch die Version V4B für die PDP 7 umgeschrieben wurde. Im ADSS arbeiten zwar alle externen Geräte im Realzeitbetrieb, aber es gibt darin keine Möglichkeit für einen Multitaskbetrieb. Deshalb mußte es für den simultanen Betrieb von mehreren Tasks erweitert werden. (In der Umgangssprache wird das ADSS als Monitor-Betriebssystem bezeichnet.)

6.0 Die Struktur des Betriebssystems

Von einem Betriebssystem für den Realzeitbetrieb wird gefordert, daß es kurze Reaktionszeiten besitzt, was praktisch bedeutet, daß möglichst zu jedem Zeitpunkt ein Interrupt bearbeitet werden kann. Dem steht aber entgegen, daß aus Sicherheitsgründen für gewisse Programmabläufe Interruptsperrern gegeben werden müssen.

Damit diese Interruptsperrern recht kurz gehalten werden können, werden logische Ebenen definiert, auf denen Programmteile unabhängig voneinander laufen und bei denen nur die Übergänge zwischen den verschiedenen Ebenen geschützt werden müssen.

Das CAMAC-Betriebssystem arbeitet auf drei logischen Ebenen, die gegeneinander synchronisiert werden. Auf der unteren Ebene werden die Ein-Ausgaben durchgeführt, die durch Interrupt von den Geräten gesteuert werden. Ausführende Programmteile sind die Device-Handler. Auf der mittleren Ebene, Zwischenebene genannt, läuft ein Formatierungsprogramm, das die FORTRAN-Ein-Ausgaben aller Geräte bearbeitet, während auf der oberen Ebene die vom Benutzer geschriebenen Tasks ausgeführt werden. (Vergl. Abb. 3.)

Auf der oberen Ebene werden die Programme (Task's) nach der Priorität abgearbeitet. Durch den Eintrag aller an- gestoßenen Tasks in eine Warteschlange ist sichergestellt, daß nach Beendigung bzw. Unterbrechung einer Task die nächste bearbeitet wird. Die Organisation dafür übernimmt der Taskkontroller.

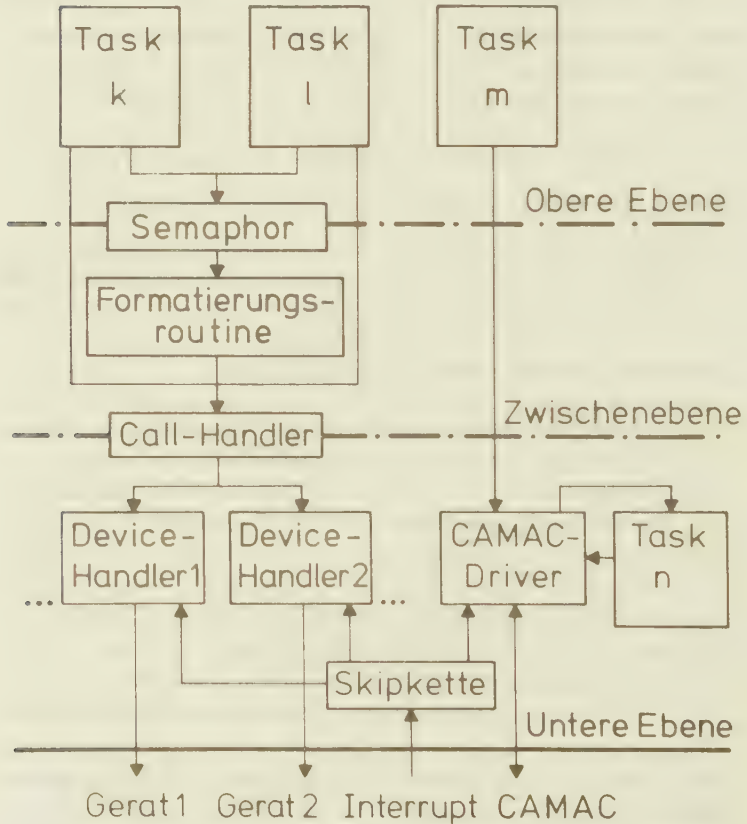


Abb. 4: Die logischen Ebenen des Betriebssystems

Bei einer Ein-Ausgabe wird die obere Ebene verlassen. Wenn es sich um eine formatierte Ein-Ausgabe handelt, wird die Zwischenebene aufgerufen. Diese wird durch ein binäres Semaphor (Belegtanzeige) geschützt, das die Zustände "belegt" und "frei" anzeigt. Während bei einem freien Formatierungsprogramm eine Task weiter bearbeitet wird, muß bei einem belegten Zustand die Task in eine Warteschlange für das Formatierungsprogramm eingeordnet werden, wobei gleichzeitig die Task aus der CPU-Warteschlange ausgetragen wird.

Der Übergang auf die untere Ebene geschieht über einen CALL, der innerhalb des Betriebssystems die Gerätezuweisung vornimmt. Es ist dabei unerheblich, ob der Aufruf von der oberen Ebene oder aus der Zwischenebene erfolgt. Es sei darauf hingewiesen, daß eine ganze Reihe von Ein-Ausgaben kein Formatierungsprogramm benötigen, wie z.B. Ein-Ausgaben auf DECTape.

Nach der Gerätezuweisung durch das Betriebssystem wird über ein Semaphor der Zustand des Gerätes geprüft. Bei freiem Gerät wird die Ein-Ausgabe angestoßen, so daß danach die Bedienung des Gerätes auf der unteren Ebene vorgenommen wird, ohne daß die oberen Ebenen berührt werden. Sollte das Gerät belegt sein, so wird die Task in eine Warteschlange auf das entsprechende Gerät eingetragen. Obwohl alle Geräte gemeinsam auf der unteren Ebene liegen, besitzt jedes Gerät eine Warteschlange, da die Geräte voneinander unabhängig arbeiten und sich nicht beeinflussen. Es ist somit möglich, daß sich die Device-Handler gegeneinander durch Interrupts unterbrechen, weil jedes Gerät seine eigene Bearbeitungsroutine besitzt. Die Priorität der Bearbeitung der Geräte wird durch die Reihenfolge in der Skipkette festgelegt.

Wenn bei Eingaben die Daten benötigt werden, wird durch einen WAIT-Aufruf vorher auf die Beendigung der Operation getestet. Solange die Eingabe noch nicht beendet ist, wird die entsprechende Task auch in die Warteschlange des Gerätes eingeordnet.

Eine Sonderstellung nimmt CAMAC ein, weil im CAMAC keine formatierten Ein-Ausgaben durchgeführt werden, und weil CAMAC mit Einzelworttransfer arbeitet. Deshalb wird ein CAMAC-Befehl, wie er als Aufruf in einem Programm steht, als von CAMAC ununterbrechbare Routine ausgeführt. Da es sich dabei um kurze Programme handelt, würde eine andere Organisation einen zu großen Verwaltungsaufwand erfordern.

Die Ununterbrechbarkeit von CAMAC ist erforderlich, weil zur Ausführung eines CAMAC-Befehls mehrere IOT-Befehle benötigt werden. Ein Interrupt von CAMAC während der Bearbeitung eines CAMAC-Befehls würde die bereits geladenen Registerinhalte zerstören, wenn das LAM-Muster gelesen wird.

Während die Standardgeräte eigene Bearbeitungsroutinen auf der unteren Ebene besitzen, werden die Bearbeitungsroutinen für ein LAM von CAMAC vom Benutzer geschrieben. Eine Task, die von einem LAM angestoßen wird, läuft auf der unteren Ebene, bis der Benutzer eine TAKE-Anweisung eingefügt hat. Erst dann wird die Task auf der oberen Ebene weiterverarbeitet. Bis dahin ist auch die Unterbrechungssperre von CAMAC gesetzt. Kurze Bearbeitungsroutinen, die keine TAKE-Anweisung enthalten, werden nur auf der unteren Ebene bearbeitet, so daß sie schnell sind und keine Betriebssystemaufrufe benötigen.

Die Fortsetzung von Tasks nach beendeter Ein-Ausgabe und die Abarbeitung der Warteschlangen der Geräte erfolgt durch zyklische Abfrage der Geräte auf beendete Ein-Ausgabevorgänge innerhalb des Displayprogramms. Sobald eine beendete Operation festgestellt worden ist, wird geprüft, ob noch eine Task auf das Gerät wartet. Wenn dies der Fall ist, wird die nächste Ein-Ausgabe angestoßen, und die entsprechende Task wird wieder in die Warteschlange der CPU eingeordnet. Bei einer WAIT-Operation wird somit die zugehörige Task ebenfalls fortgesetzt. Da dabei aber noch

keine Ein-Ausgabe angestoßen wurde, wird geprüft, ob noch eine weitere Task auf das Gerät wartet.

In der Arbeit /28/ wurde ein Verfahren entworfen, das die Abarbeitung der Warteschlangen den Device-Handlern überträgt und damit die zyklische Abfrage der Geräte auf beendete Ein-Ausgaben erübrigt. Bei den derzeitigen Anwendungen ist diese Umstellung nicht erforderlich, da genügend Bearbeitungszeit zur Verfügung steht.

6.1 Der Taskkontrolller

Im Monitor-Betriebssystem existiert eine allgemeine Schnittstelle zwischen den Ein-Ausgabeeinweisungen (Systemmacros) der Benutzerprogramme und den ausführenden Device-Handlern, der sogenannte Call-Handler. Da alle Ein-Ausgabeeinweisungen diese Schnittstelle benutzen, wurde der Call-Handler um einen Taskkontrolller erweitert.

Der Taskkontrolller hat die Aufgabe, die prioritätsgesteuerte Abarbeitung der Tasks zu koordinieren. Jede Task erhält dazu bei ihrer Einrichtung einen Taskkontrollblock, in dem der Zustand der Task geführt wird. Die Abbildung 4 zeigt den Aufbau des Blocks, der aus 7 Zellen besteht.

Nachdem beim Start einer Task die Besetztanzeige geprüft und gleichzeitig gesetzt wurde, wird der zugehörige Taskkontrollblock über die zwei Verkettungszeiger entsprechend der Priorität in die Warteschlange der CPU eingebunden. Der Taskkontrolller entnimmt außerdem aus dem Kontrollblock die Taskanfangsadresse und legt sie als Unterbrechungsadresse ab. Der AC-Speicher wird beim Starten nicht gesetzt, er wird nur für eine Unterbrechung benötigt. Die Bearbeitung der Task hängt danach von dem Eintrag in der Warteschlange und den möglichen Wartezuständen aller Task für E/A-Vorgänge ab.

Zelle	Inhalt	Erläuterung
1.	Besetztanzeige	frei: -1; besetzt: +1
2.	Task Nr.	1 bis 1000 = Priorität
3.	Vorwärtszeiger	Verkettungszeiger
4.	Rückwärtszeiger	Verkettungszeiger
5.	Taskanfangsadresse	
6.	Unterbrechungs- adresse	für Wartezustand
7.	AC-Speicher	für Wartezustand

Abb. 4: Aufbau eines Taskkontrollblocks

Zum Lauf der Task erhält die CPU aus dem Taskkontrollblock die Unterbrechungsadresse sowie, wenn es vom Programm notwendig ist, den AC-Inhalt.

Trifft eine Task bei ihrer Bearbeitung auf eine Ein-Ausgabe, so wird sie über den Call-Handler dem Taskkontroller übergeben. Der Taskkontroller prüft zunächst, ob das angesprochene Gerät belegt ist. Wenn es keine E/A-Anweisung bearbeitet, wird die folgende E/A-Anweisung direkt dem Device-Handler übergeben. Ist das E/A-Gerät aber belegt, wird die laufende Task unterbrochen, und der zugehörige Taskkontrollblock wird aus der CPU-Warteschlange ausgetragen und in die Warteschlange für das betreffende Gerät eingetragen. Dazu besteht für jedes Gerät eine eigene Warteschlangenorganisation.

Während der Wartezeit einer Task auf einen E/A-Vorgang kann eine Task niederer Priorität laufen. Z.Z. ist der simultane Betrieb von 1000 Tasks möglich, wobei 1000 keine prinzipielle Grenze darstellt, sondern nur eine Begrenzung der Tasknummern ist, die aber für die derzeitigen Anwendungsprobleme sicher ausreichend ist.

6.2 Der Startmechanismus

Eine Task kann auf drei verschiedene Arten gestartet werden:

1. vom Fernschreiber über einen Startbefehl,
2. von einer laufenden Task über eine Startanweisung, oder
3. durch einen Interrupt von CAMAC.

Jede dieser drei Möglichkeiten besetzt durch einen Eintrag in die erste Zelle des Taskkontrollblocks die Task und bindet den Kontrollblock in die Warteschlange der CPU entsprechend der Priorität ein.

Während in den ersten beiden Fällen das System in einem definierten Zustand im Betriebssystem ist, dem Taskstartprogramm, das ein Teil des Taskwechselprogramms ist, und deshalb auch sofort ein Taskwechsel vorgenommen werden kann, muß der Start einer Task durch ein CAMAC-Interrupt in zwei Stufen durchgeführt werden. Da ein Interrupt von CAMAC in jedem Programmmzustand erfolgen kann, das System ist stets unterbrechbar, wäre es möglich, daß das Programm in einem "kritischen" Programmteil, z.B. in einem FORTRAN-Hilfsprogramm, arbeitet.

Um auch in solchen Zuständen einen CAMAC-Interrupt bearbeiten zu können, wird in der ersten Stufe die bearbeitende Task direkt vom Interrupt ohne Einwirkung des Taskkontrollers angestoßen. In dieser Stufe muß vordringlich erst einmal die Ursache des Interrupts beseitigt werden. Der Benutzer hat die Aufgabe, im Taskkopf der zuständigen Task diese Arbeit zu erledigen. Weil in dieser Situation auch kein weiterer Interrupt von CAMAC bearbeitet werden kann, ist der Interrupt von CAMAC auch abgeschaltet. Danach ergeben sich für den Benutzer zwei Möglichkeiten, die von der Aufgabenstellung abhängen:

1. Die Bearbeitungstask ist sehr kurz und benötigt keine arithmetischen Operationen. Dann braucht sie überhaupt nicht dem Taskkontroller übergeben zu werden. Nachdem die Ursache des Interrupts beseitigt ist, wird die Task durch eine RETURN-Anweisung beendet. Damit wird auch der Interrupt von CAMAC wieder freigegeben und das unterbrochene Programm wird fortgesetzt.
2. Die Task ist zeitlich zu lang, und/oder sie benötigt arithmetische Operationen. Dann muß genau wie im ersten Fall die Ursache des Interrupts beseitigt werden, aber der Benutzer übergibt mit der TAKE-Anweisung die Task der Tasksteuerung. Dadurch wird sie in die Warteschlange der CPU entsprechend der Priorität eingetragen, aber, auch wenn die Priorität hoch genug wäre, wird sie nicht sofort gestartet. Nach dem Eintrag in die Warteschlange schließt die TAKE-Anweisung mit der Einschaltung des Interrupts von CAMAC ab und die CPU kehrt in das durch den CAMAC-Interrupt unterbrochene Programm zurück. Erst wenn dieses Programm an eine Taskwechselstelle kommt, wird der Taskwechsel vollzogen.

Der Benutzer hat stets die Aufgabe zu verhindern, daß eine so gestartete Task nicht wieder gestartet wird, bevor sie beendet ist. Dies läßt sich im allgemeinen leicht realisieren. Sollte dennoch ein 2. Interrupt erfolgen, wird er ignoriert.

6.3 Der Taskwechsel

Ein Taskwechsel kann prinzipiell bei jedem E/A-Vorgang erfolgen. Zu diesem Zeitpunkt wird über den Call-Handler der Taskkontroller aufgerufen. Er prüft neben dem angesprochenen Gerät auch alle anderen Geräte auf beendete Ein-Ausgaben. Hat ein Gerät eine Ein-Ausgabe beendet, so wird es freigegeben, falls keine andere Task darauf wartet.

Wenn in der Warteschlange für das Gerät eine Task eingetragen ist, so wird die zurückgestellte Ein-Ausgabe gestartet, und die Task wird aus der Warteschlange für das entsprechende Gerät ausgetragen und wieder in die CPU-Warteschlange eingetragen.

Fortgesetzt wird danach die jeweils höchstprioritäre Task in der CPU-Warteschlange, so daß damit auch eine Task, die von CAMAC gestartet wurde, bearbeitet wird, wenn ihre Priorität nur hoch genug ist.

Beendet werden Tasks durch den Austrag ihres Taskkontrollblocks aus der Warteschlange der CPU nach der entsprechenden Anweisung im Programm. Gleichzeitig wird die Besetztanzeige im Taskkontrollblock zurückgesetzt.

Solange keine Task in der CPU-Warteschlange steht, läuft das Displayprogramm für Spektren im Hintergrund. Auch in diesem Programm werden alle Geräte zyklisch auf beendete E/A-Vorgänge geprüft und neu in die CPU-Warteschlange eingetragene Tasks werden fortgesetzt. Das Displayprogramm ist nicht als Task aufgebaut.

6.4 Das Taskzustandsdiagramm

Für die Zustände der Tasks und die möglichen Übergänge zwischen ihnen läßt sich ein Zustandsdiagramm aufstellen. (Vergl. Abb. 5.)

Der Übergang von einer nicht existierenden Task zum Zustand bekannt geschieht durch den Ladevorgang. Dagegen läßt sich eine existierende Task nicht wieder löschen, dies kann nur indirekt über einen Neustart erfolgen.

Aus dem Zustand bekannt kann eine Task über die drei Startmöglichkeiten in den Zustand Warten auf CPU kommen. Beim CAMAC-Interrupt ist es möglich, daß die Task gestartet wird, ohne daß sie in die Warteschlange der CPU eingeordnet wird und somit wieder im gleichen Zustand endet.

Die jeweils höchstpriorie Task wird aus dem Zustand Warten auf CPU in laufend versetzt. Aus diesem Zustand lassen sich alle anderen Zustände, ausgenommen nicht existent, erreichen.

Solange ein Ein-Ausgabegerät belegt ist, wird die Task in die Warteschlange des E/A-Gerätes eingeordnet und damit in den Zustand Warten auf E/A versetzt. In den Zustand bekannt gelangt eine Task durch die Beendigung oder durch einen Fehlerausgang. Sich selbst kann eine Task aus dem Zustand laufend durch den Start einer höher-priorien Task verdrängen.

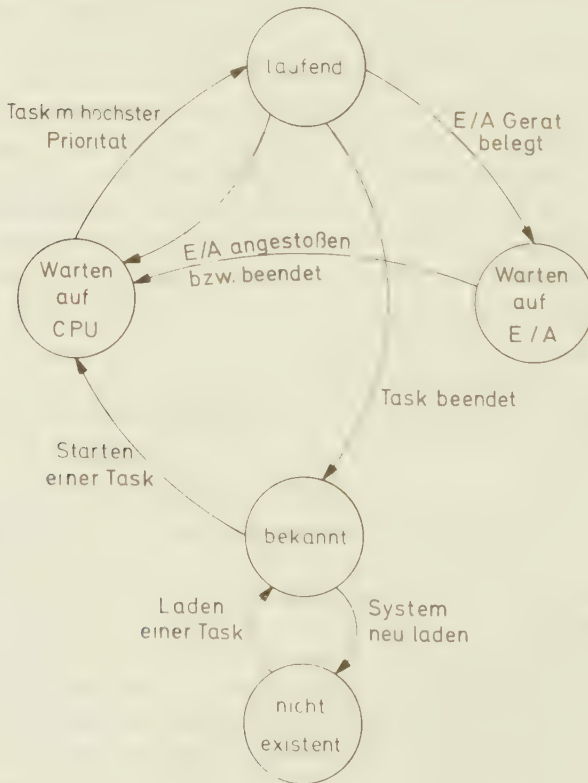


Abb. 5: Das Taskzustandsdiagramm

Eine Task im Zustand Warten auf E/A wird nach einem beendeten E/A-Vorgang des entsprechenden Gerätes fortgesetzt und damit in den Zustand Warten auf CPU versetzt.

6.5 Die CAMAC-Interruptverarbeitung

Abgesehen von kurzen Routinen beim Taskwechsel ist der Interrupt stets eingeschaltet. Weil die PDP 7 keine prioritätsgesteuerte Interruptverarbeitungshardware besitzt, ist damit auch der Interrupt von CAMAC stets freigegeben, wenn die Anweisung INITC durchlaufen und kein CLOSEC gegeben wurde.

Nach dem Interrupt durchläuft die Interruptroutine die Skipkette bis zum CAMAC-Skip, der an 4. Stelle steht. Danach erfolgt die Bearbeitung im CAMAC-Devicedriver.

Als erstes wird die CAMAC-Interrupt abgeschaltet und der Interrupt aller anderen Geräte wieder freigegeben. Danach wird das LAM-Muster gelesen und ausgewertet. Dazu existiert im CAMAC-Handler eine Sprungliste für 24 Sprungadressen, entsprechend der Anzahl der möglichen LAM's im LAM-Muster. Die Sprungliste wird durch die DO-Anweisung mit der Adresse für die anzusteuernde Task beim entsprechenden LAM gesetzt.

Bei einem CAMAC-Interrupt wird über die Liste die zum gelesenen LAM zugehörige Task gestartet. Wenn mehrere LAM gleichzeitig eintreffen, wird das höchstpriorie LAM (niedrigste Nummer) zuerst bearbeitet. Erst nach der TAKE-Anweisung bzw. bei kürzeren Tasks nach deren Beendigung wird der CAMAC-Interrupt wieder eingeschaltet. Solange läuft die Task als Teil der Interrupt-Service-Routine von CAMAC.

Nachdem ein Interrupt von CAMAC bearbeitet wurde, kehrt die Steuerung wieder in den CAMAC-Handler zurück, um den CAMAC-Interrupt wieder einzuschalten. Danach kann ein nächster LAM bearbeitet werden, der auch wieder über die Skip-Kette läuft.

Damit dieser Mechanismus nicht vollkommen bei ADC's durchlaufen wird, die auch z.B. koinzident arbeiten könnten, wird für die ersten drei LAM's der Stationen 1, 2 und 3 noch eine Zwischenspeicherung des LAM-Musters vorgenommen. Das bedeutet, daß, wenn ein LAM der Station 1 auftritt, nach dessen Bearbeitung sofort noch die LAM-Nummern 2 und 3 untersucht werden, ob sie auch gesetzt sind. Erst wenn alle drei bearbeitet sind, wird der CAMAC-Interrupt wieder eingeschaltet.

In Fehlersituationen, bei denen LAM's von Stationen kommen, denen keine Taskadresse in der Sprungliste zugeordnet wurden, wird automatisch ein DISABL mit der LAM erzeugenden Stationsadresse und der Subadresse A(0) durchgeführt. Außerdem wird die LAM-Nummer ausgedruckt.

6.6 Der CAMAC-Devicedriver

Der CAMAC-Kontroller wird über einen Devicedriver bearbeitet, der einen ähnlichen Aufbau hat wie die Device-Handler des Monitor-Betriebssystems. Da jedoch seine Funktionen nicht mit denen der Standarddevice-Handler identisch sind, wurde verzichtet, den Driver über eine logische Gerätenummer anzusprechen und ihn als Pseudohandler ins Betriebssystem zu integrieren. Er arbeitet als Unterprogramm. Der Driver übernimmt zwei Aufgaben:

1. Er sorgt für die Ausführung von CAMAC-Funktionen und
2. er bearbeitet die Interrupts von CAMAC.

Behandlung der CAMAC-Funktionen:

Der Aufruf des Drivers erfolgt über den globalen Unterprogrammsprung CCA, dem ein Funktionscode nachfolgt. Danach können je nach Funktion bis zu drei Worten als weitere Argumente folgen, so daß sich folgende Form ergibt:

JMS*	CCA	/ Unterprogrammsprung
nr		/ Funktionsnummer
a ₁		/ Argument 1
a ₂		/ Argument 2
a ₃		/ Argument 3

Weil nur Einzelworttransfers bei Daten zulässig sind und diese meist binäre Information übertragen, werden auch nicht die fünf Datentypen der Übertragung im Monitorsystem verwendet.

Durch diesen Aufbau ist es möglich, alle CAMAC-Funktionen über Macros zu definieren. Im Anhang C ist eine vollständige Liste der Definitionen angeführt. An dieser Stelle soll nur ein Beispiel erläutert werden.

Zum Schreiben eines CAMAC-Registers ist folgender Macro definiert, der die CAMAC-Funktion F(16) anspricht:

```
. DEFIN    WRITEC, NA,K
JMS*      CCA
10                                     /Funktionsnummer
NA                                     /Adresse der CAMAC-Adresse
407                                              /CAMAC-Funktion
K                                     /Adresse für die Information
.ENDM
```

Die erste Zelle nach dem Unterprogrammsprung enthält eine von acht Funktionsnummern, die in der Tabelle mit ihrer Bedeutung aufgeführt sind. Die CAMAC-Funktion ist bereits in einer Form codiert, in der sie sofort im CAMAC-Kontroller verwendet werden kann. NA und K sind Adressen, die auf die CAMAC-Adresse bzw. auf die zu übertragende Information zeigen.

Der Macro-Aufruf hat die gleiche Form wie die Schreib-anweisung im CoAAS.

```
WRITEC    NA,K
.
.
.
NA         na                                     /CAMAC-Adresse
K          0                                     /Schreibinformation
```

wobei auch in NA die Adresse entsprechend codiert sein muß.

Funktionsnummer	Funktionsbeschreibung
1	Schalte CAMAC-Interrupt ein
2	Schalte CAMAC-Interrupt aus
3	Datenlose CAMAC-Funktion
4	Transfer zwischen zwei CAMAC-Registern
5	Starte Analyse vom ADC
6	Stoppe Analyse vom ADC
10	CAMAC-Lesefunktion
11	CAMAC-Schreibfunktion

Tabelle 8: Funktionsnummern der Macros

Die CAMAC-Interruptverarbeitung erfolgt ebenfalls im CAMAC-Driver. In ihm wird die Interruptverarbeitung durchgeführt, wie sie in 6.5 beschrieben wurde. Es ist noch zu erwähnen, daß der INITC mit der Funktion (1) der Tabelle 8 den CAMAC-Skip in der Skipkette des Monitor-Systems absetzt.

6.7 Die Kernspeicherbelegung

Damit 8 K-Kernspeicher für Daten zur Verfügung stehen, läuft das CoAAS unter einem 16 K-Monitorsystem auf der PDP 7. Durch die Verwendung des 16 K-Monitors kann das CoAAS auch neu geladen werden, ohne daß die Daten, die in dem restlichen 8 K des 24 K großen Kernspeichers gehalten werden, zerstört werden.

Entsprechend des Aufbaus des Monitorsystems liegt am obersten Ende des für Programme verwendeten Kernspeichers der Loader, während im unteren Teil der residente Monitor steht. Das CoAAS benötigt ca. 12 K, so daß zusammen mit dem residenten Monitor ca. 13,5 K Kernspeicher belegt werden. Damit verbleiben für die Aufnahme von Tasks 2,5 K Kernspeicher.

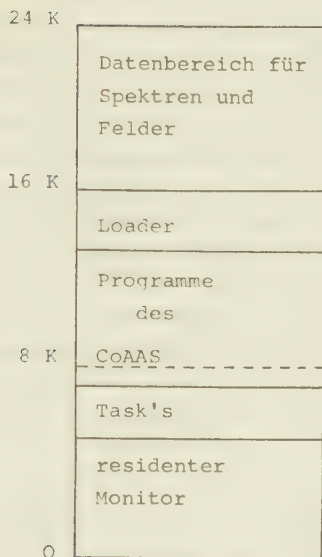


Abb. 6: Kernspeicherbelegung

Die Verwendung des 16-K-Monitors ist besonders für Experimente mit langer Akkumulationsdauer wichtig. Dadurch kann bei Experimentunterbrechungen, wie auch bei eventuellen Ausfällen, das Programm restauriert werden, ohne daß Daten verloren gehen.

6.8 Der Anschluß der FORTRAN-Unterprogramme

Für das Laufzeitpaket wurde eine Reihe von Unterprogrammen aus der FORTRAN-Bibliothek verwendet. Dazu gehören die Programme zur formatierten Ein-Ausgabe, die vollständig übernommen wurden, Programme für die Floatingpointarithmetik sowie für die Integermultiplikation und -division und Hilfsprogramme für den Zugriff auf dimensionierte Variable.

Alle benötigten Unterprogramme werden bereits mit dem System geladen, so daß ihre Adressen bekannt sind und sofort vom Übersetzerprogramm eingesetzt werden können. Weil der Aufruf und das Programm in verschiedenen Kernspeicherbänken liegen können (und meist auch liegen), werden die Unterprogramme stets indirekt angesprungen. Dazu werden die Adressen der Unterprogramme beim Start des Systems im Variablenpuffer abgesetzt, der in der gleichen Bank wie die übersetzten Programme liegt.

Beispiel:

Der Aufruf des Divisionsprogramms:

Bank n

Programmpuffer

```

:
:
LAC      IVAR1    / Hole Dividend
JMS*     AIDIV    / Sprung ins Unterprogramm
LAC      IVAR2    / Hole Divisor
:
:

```

Variablenpuffer

```

:
:
AIDIV     IDIVIA      / Adresse des Divisionsprogramms
:
:
IVAR1     n1         / Variable 1
:
IVAR2     n2         / Variable 2
:
:

```

Bank m

```

:
:
IDIVIS    Ø           / Divisionsunterprogramm
:
:
JMP*      IDIVIS     / Rücksprung
:
:

```

6.9 Dynamische Kenngrößen des Systems

Zur Bestimmung der Leistungsfähigkeit eines Systems, d.h. von Hard- und Software gemeinsam, sind Laufzeiten von zentralen Routinen wichtig.

Sie werden hier zusammengestellt:

1. Die Interruptbearbeitungszeit

Die Interruptbearbeitungszeit T_I ist die Zeit, die vom Auftreten des Interrupts bis zu seiner Verarbeitung durch das zugehörige Programm vergeht.

Weil sowohl das Gerät als auch die LAM-Nummer softwaremäßig ermittelt werden, ist diese Zeit relativ lang, verglichen mit hardwareunterstützten Interruptbearbeitungen. Für CAMAC hängt die Interruptbearbeitungszeit T_I^C auch von der LAM-Nummer n ab:

$$T_I^C = 81 \mu\text{sec} + n * 7 \mu\text{sec}$$

Für die drei höchstpriorären LAM's (niedrigste Stationsnummern) vermindert sich die Antwortzeit bei gleichzeitigen LAM's von allen drei Stationen, da nach dem Auftreten des ersten LAM's auch gleich die beiden folgenden geprüft werden. Dadurch ergibt sich eine kürzere gemeinsame Interruptbearbeitungszeit T_I^{1-3} für alle drei LAM's:

$$T_I^{1-3} = 81 \mu\text{sec} + 2 * 10,5 \mu\text{sec}.$$

Dies ist besonders für koinzidente Messungen über CAMAC wichtig, da sich dadurch der Durchsatz erhöht.

Zum Vergleich beträgt die Interruptverarbeitungszeit für den Nuclear Data ADC mit speziellem Interface

$$T_I^{\text{ADC}} = 19 \mu\text{sec}.$$

2. Bearbeitungszeiten

Die Bearbeitungszeit T_B ist die Zeit, die ein Programm nach dessen Start benötigt bis der Auftrag ausgeführt ist. In diesem Fall ist es die Zeit vom Start einer Task von CAMAC bis zu ihrer Beendigung. Diese Zeit hängt vom Benutzerprogramm ab, so daß dazu keine speziellen Angaben gemacht werden können. Es lassen sich aber Zahlen über die Bearbeitungszeiten der ADC's angeben, da diese über AssemblerROUTINEN bearbeitet werden.

Die Bearbeitungszeit T_B^C eines ADC's über CAMAC beträgt:

$$T_B^C = 30 \mu\text{sec}$$

und für die ADC's mit eigenem Interface

$$T_B^{\text{ADC}} = 43 \mu\text{sec}.$$

Für Abschätzungen ist auch die Bearbeitungszeit für die Ausführung einer CAMAC-Lese-Schreibanweisung interessant. Sie beträgt

$$T_B^{L/S} = 120 \mu\text{sec}.$$

3. Verwaltungszeiten

Unter einer Verwaltungszeit T_V versteht man die Zeit, die vom Betriebssystem benötigt wird.

Die Verwaltungszeiten gehen stets den Programmlaufzeiten verloren und sie sollten deshalb möglichst gering gehalten werden. Für das System sind drei Verwaltungszeiten wichtig.

a) Die Interruptrücksprungzeit T_R^C von CAMAC

$$T_R^C = 23 \mu\text{sec}$$

b) Die Übergabezeit T_U einer Task an das Betriebssystem

$$T_U = 175 \mu\text{sec}$$

c) Taskwechselzeit

$$T_W$$

$$T_W = 85 \mu\text{sec}.$$

4. Antwortzeit

Die Antwortzeit T_A ist die Zeit, die vom Auftreten des Interrupts bis zu dessen Bearbeitung benötigt wird. Für die Bearbeitung eines ADC's in CAMAC beträgt die Antwortzeit

$$T_A^C = T_I^C + T_B^C + T_R^C$$

$$T_A^C = 147 \mu\text{sec}$$

Für die Bearbeitung von drei ADC's in CAMAC beträgt sie:

$$T_A^{1-3} = T^{1-3} + 3 T_B^C + T_R^C$$

$$T_A^{1-3} = 168 \mu\text{sec}$$

Zum Vergleich beträgt die Antwortzeit für den ADC mit speziellem Interface

$$T_A^{\text{ADC}} = T_I + T_B$$

$$T_A^{\text{ADC}} = 62 \mu\text{sec}$$

Für die Abschätzung der Antwortzeit T_A^T einer Task ohne Übergabe aus Betriebssystem gilt:

$$T_A^T = T_I^C + T_B + T_R^C$$

wobei T_B die Bearbeitungszeit der Task ist.

Bei einer Task mit einer Übergabe aus Betriebssystem kommt noch ein T_U dazu.

5. Der Durchsatz

Aus den Antwortzeiten läßt sich der Durchsatz D ermitteln:

$$D = \frac{1}{T_A}$$

Für die Analyse über die ADC's mit CAMAC liegt der Durchsatz zwischen 68 kHz für einen ADC bzw. mehrere nicht koinzident betriebene ADC's gemeinsam und 59 kHz für 3 koinzident betriebene ADC's.

Demgegenüber liegt der Durchsatz für den Nuclear Data ADC bei ca. 160 kHz.

Da das Bildschirmprogramm als Hintergrundprogramm läuft, läßt sich die Auslastung des Rechners rein qualitativ durch die Güte des Bildes erkennen. Solange kein größeres Flackern, oder anders ausgedrückt, kein langsames Zeichnen des Bildes erfolgt, ist die Auslastung noch nicht im Grenzbereich.

6.10 Fehlerbehandlung

Die Fehler lassen sich in zwei Klassen einteilen, in die syntaktischen und in die Laufzeitfehler. Weil sich das Laufzeitpaket aus den FORTRAN-Unterprogrammen und den eigenen Programmen zusammensetzt, unterteilen sich die Laufzeitfehler nochmals in zwei Gruppen.

1. Syntaktische Fehler

Die syntaktische Prüfung jeder Anweisung wird bei der Eingabe vorgenommen. Fehlerhafte Anweisungen werden verworfen, d.h. sie werden entweder nicht ausgeführt, oder es wird kein Übersetzter Code abgesetzt. Außerdem werden sie im Taskerstellungsmodus auch nicht auf DECTape abgespeichert. Ein syntaktischer Fehler wird durch die Fehlermeldung

FEHLER n

gemeldet. Bei der Übersetzung vom DECTape wird die Übersetzung an der Fehlerstelle abgebrochen. *

2. Laufzeitfehler

Unter den Begriff Laufzeitfehler sind alle Fehler zusammengefaßt, die zur Laufzeit auftreten. Darunter fallen alle semantischen Fehler. Sie führen zum Abbruch

der entsprechenden Anweisung bzw. zum Beenden der Task. Als einziger Laufzeitfehler wird der Doppelstart einer Task nicht gemeldet. Dieser Fehler muß vom Benutzer verhindert werden. Er tritt meist dann auf, wenn eine Task bereits wieder gestartet werden soll, bevor der vorhergehende Meßwert abgearbeitet worden ist. In diesem Fall liegt eine Überlastung des Rechners vor. Zusätzliche Fehlermeldungen an dieser Stelle würden die Verarbeitungszeit nur vergrößern.

Um trotzdem noch eine Verarbeitung zu gewährleisten, ohne daß Meßwerte verloren gehen, wäre der Aufbau einer Warteschlange für die Taskstarts notwendig. Erfahrungsgemäß bringt dies nur selten einen Nutzen, verglichen mit dem hohen Aufwand.

3. FORTRAN-Fehler

Fehler, die in FORTRAN-Routinen zur Laufzeit auftreten, werden von den FORTRAN-Fehler Routinen bearbeitet und liefern eine

IOPS n

Meldung, die zum Abbruch des Programms führt. Diese Fehler treten bei Bereichüberschreitungen in den Arithmetik-Routinen auf. Weiterhin sind die Fehler bei ungetesteten Programmen in den Formatierungsroutinen möglich. Falls Geräte abgeschaltet wurden, kann nach einer

IOPS 4

Meldung mit

CNTRL R

nach Einschalten des entsprechenden Gerätes fortgesetzt werden.

7. Der Einsatz des Systems im Experiment

Das CoAAS wurde zuerst für Messungen von T_C -Änderungen an verschiedenen Substanzen nach Tieftemperaturbestrahlung eingesetzt. Bei Messungen dieser Art wurden bislang die Meßdaten analog auf X - Y - bzw. X - t-Schreibern aufgenommen. Teilweise wurden die Meßwerte auch manuell abgelesen und notiert, was abgesehen von der anstrengenden Arbeit bei kurzen Zeitintervallen auch recht fehleranfällig war. Auch die anschließende Auswertung nahm oft mehrere Monate in Anspruch. Durch die Aufzeichnung der Maßdaten on line über die PDP 7, die über ein benutzergeschriebenes Programm erfolgt, konnte neben einer rationelleren Arbeitsweise bei der Meßwerterfassung auch eine wesentliche Genauigkeitssteigerung erreicht werden. Außerdem liegen die Meßdaten computergerecht vor, so daß sie ohne Mühe in anschließende Auswerteprogramme übernommen werden können.

7.1 Die Meßanordnung

Weil der Temperaturverlauf relativ schlecht zu beeinflussen ist, wurden die Messungen zyklisch durchgeführt und die zusammengehörigen Meßwerte in Listen gespeichert. Der Temperaturverlauf der Probe wurde mit Hilfe eines Ge-Widerstandsthermometers über ein Dana-Voltmeter gemessen, das wiederum über ein Telefunkteneingaberegister MS P12 1230/1 vom Rechner über CAMAC ausgelesen wurde. Der Spannungsabfall der Probe wurde mit einem Siemens-CAMAC-Voltmeter gemessen. Die Meßanordnung zeigt Abb. 7.

Da die Wertebereiche der Spannungen mehrere Zehnerpotenzen überstreichen können und im niedrigsten Bereich unter dem mV-Bereich liegen, konnte die sonst am Tandem übliche Meßwertübertragung zum Rechner hin nicht erfolgen. Stattdessen mußten die Meßwerte am Ort des Experiments gemessen und digital übertragen werden. Dazu wurde ein CAMAC-Crate in der Nähe des Experiments installiert und über ein 60 m-Kabel am Rechner angeschlossen. Um die Übertragung über ein langes Kabel zu ermöglichen, mußte die Zykluszeit der

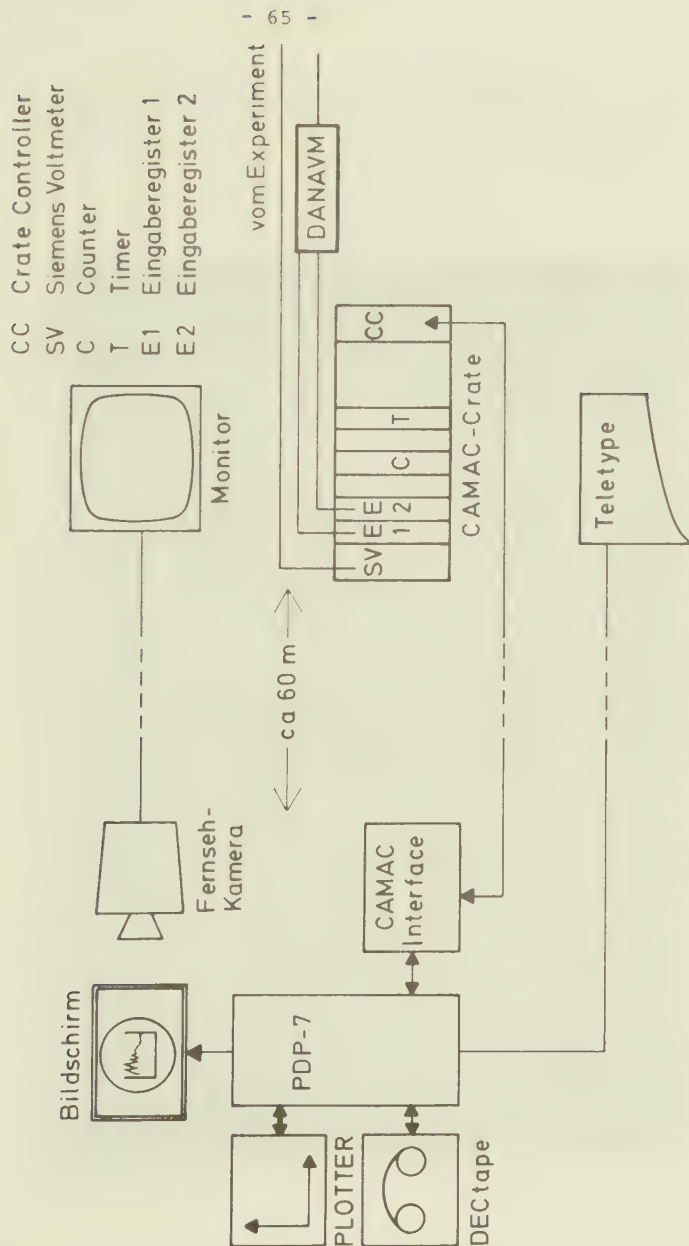


Abb. 7: Übersichtsbild des Experimentaufbaus

PDP 7 für den CAMAC-Skip CCSIL um ca. 500 nsec verlängert werden, um die Kabelllaufzeiten zu berücksichtigen /26/.

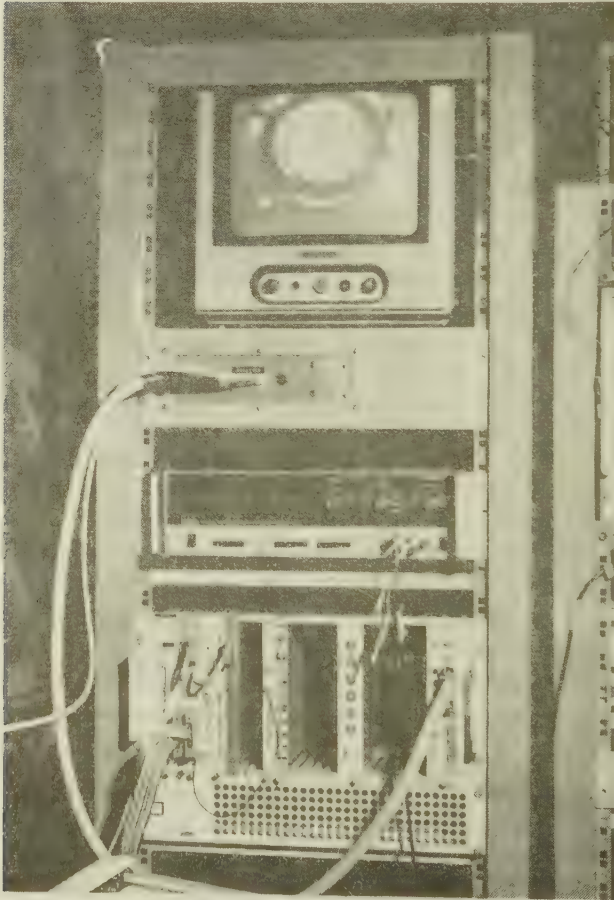


Abb. 8: Meßplatz mit CAMAC-Crate und Monitor

Die Vorgabe der Meßzeitintervalle erfolgt über einen CAMAC-Vorwahlzähler FHC 1301, der von einem Timer FHC 1303 angesteuert wurde und nach erreichter Vorwahl einen Interrupt lieferte. Bei der Zeitvorwahl muß die Verschlüsselungszeit der beiden Voltmeter berücksichtigt werden.

Für die Messung ist auch die Kontrolle der Meßdaten notwendig. Dazu wurde mit Fernsehübertragung das Bild des Bildschirms an der PDP 7 zu einem Monitor am Meßplatz übertragen. Abb. 8 zeigt den Meßplatz.

7.2 Das Meßprogramm

Zur Meßdatenerfassung wurden 6 Tasks (vergl. Anhang A) geschrieben, die eine zyklische Abfrage der beiden Voltmeter vornehmen und die Daten in 3 Felder abspeichern. Nachdem die Felder gefüllt sind, können die Daten auf DECTape abgespeichert werden, während die Datenaufnahme in einen zweiten Satz von Feldern fortgesetzt wird. Weil nicht alle Meßdaten für eine Abspeicherung interessant sind, wird die Abspeicherung von dem Experimentator vorgenommen, der die Messung am Bildschirm verfolgen kann.

Die Aufgaben der Tasks

1. Die Starttask: TASK 1

Von dieser Task werden die verwendeten CAMAC-Module eingeschaltet, sowie das Vorwahlregister für das Meßintervall gesetzt.

2. Die Task zum Start der Messung: TASK 2

Nachdem die Meßzeit abgelaufen ist, wird eine Task gestartet, die die Messung von Temperatur und Restspannung durch das DANA-Voltmeter und das CAMAC-Voltmeter startet.

3. Zwei Tasks zum Auslesen der Meßwerte: TASK 4 und 5

Da beide Voltmeter unterschiedliche Verschlüsselungszeiten besitzen und zusätzliche Zeiten durch Meßbereichsumschaltungen auftreten können, wurde für jedes Voltmeter getrennt eine Task zum Auslesen des Meßwertes geschrieben. Die Tasks übernehmen gleichzeitig die Meßbereichsauswertung, so daß die Daten als Integerzahlen in die Datenbereiche abgespeichert werden können. Dadurch kann der Experimentator sofort seine Meßergebnisse überprüfen. Die Weiterschaltung des Abspeicherindex im Feld wird danach für beide Tasks synchronisiert und gleichzeitig der Vorwahlzähler für das nächste Meßintervall gestartet.

Nachdem 1024 Meßpunkte übernommen wurden, wird auf den Fernschreiber die Meldung ENDEL ausgedruckt und gleichzeitig starten die Tasks eine neue Messung in einen zweiten Satz von Datenbereichen.

4. Abspeichern der Meßwerte: TASK 7 und 8

Über die TASK's 7 bzw. 8 können die Meßwerte des ersten bzw. zweiten Satzes von Feldern abgespeichert werden. Diese Tasks werden über den Fernschreiber durch die Befehle

S 7

bzw.

S 8

gestartet.

7.3 Messungen

Mit Hilfe der Programme wurden Meßreihen zur Ermittlung der T_C -Änderung von Nb_3Sn nach Tieftemperaturbestrahlung mit 25 MeV $^{16}O^{4+}$ -Ionen nach verschiedenen Bestrahlungsdosen durchgeführt. Eine typische Messung ist in den Abbildungen 9 und 10 dargestellt. Abb. 9 zeigt den Spannungsabfall über der Probe mit einer zeitlichen

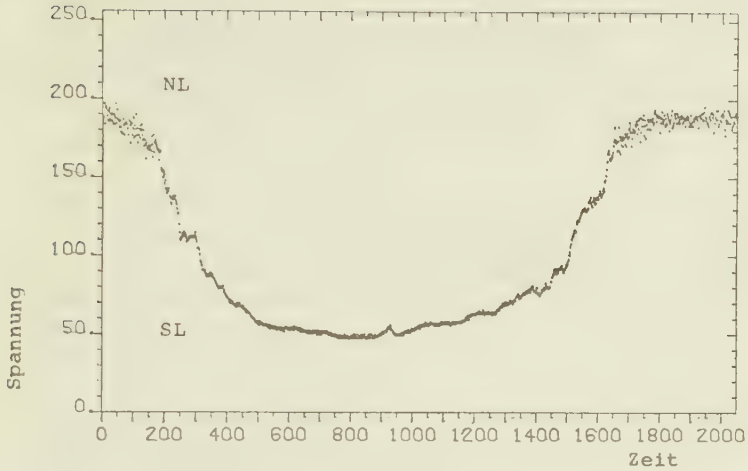


Abb. 9: Spannungsabfall über der Probe
NL: Normalleitung
SL: Supraleitung

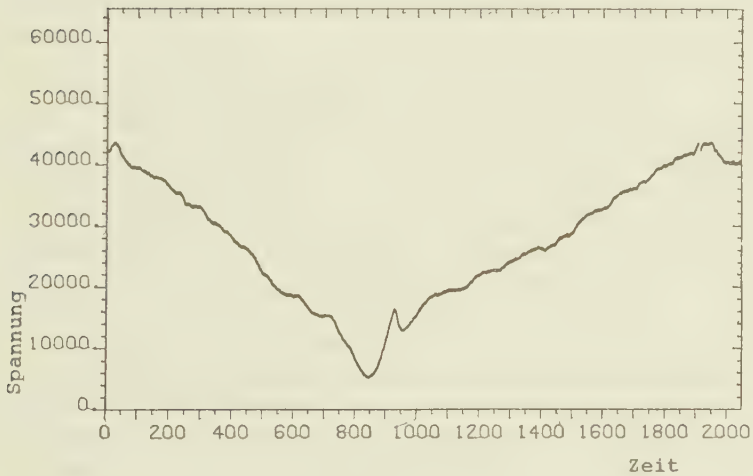


Abb. 10: Widerstand des Ge-Temperaturfühlers

Schrittweise von 0,1 sec. Zeitsynchron dazu wurde der Widerstand des Temperaturmeßfühlers aufgenommen. Den Einheiten auf den Ordinaten entsprechen die kleinsten Anzeigedigits der ausgelesenen Meßinstrumente.

8. Schlußbemerkung

Außer zur Steuerung und direkten Datenaufnahme von Experimenten läßt sich das System auch für die Erstellung von Devicedrivers für CAMAC-Interface und zur Auswertung von Messungen einsetzen.

Im Anhang B ist ein Programm für den Übertrag von Spektren von einem Canberra-Vielkanalanalysator über ein auf die V24 Schnittstelle umgebautes Teletypewriter Interface 7061 von Nuclear Enterprises LTD, das mit einer Übertragungsfrequenz von 4800 Bd arbeitet, angefügt.

Den Einsatz des Systems für spezielle Auswerteprogramme läßt der beschränkte Kernspeicherplatz nicht zu, da diese Programme meist recht umfangreich sind. Deshalb lassen sich nur kleinere Aufgaben on line lösen. Geeignet dazu sind z.B. auch Protokollieren von Messungen.

Aus Platzgründen mußte auf die Aufnahme der Standard-Funktion, wie sie in FORTRAN zur Verfügung stehen, verzichtet werden, die auch für FIT-Programme unerlässlich sind.

Als Lösung des Platzproblems bietet sich der Aufbau eines Overlaysystems an, ähnlich wie im AAS /27/, das allerdings auch in der höheren Programmiersprache zu bearbeiten sein müßte. Dabei tritt das Problem von lokalen und globalen Variablen auf, das wiederum zusammen mit einem erweiterten System wegen der zusätzlichen Verwaltung einen größeren Kernspeicher benötigen würde.

Für den Einsatz des Systems zur Experimentsteuerung wäre noch eine BIT-Bearbeitung wünschenswert, insbesondere UND- und ODER-Funktionen für die Zusammensetzung und die Analyse von Daten. Neben einer erleichterten Programmierung könnten damit die Programme auch teilweise effektiver arbeiten.

Literaturverzeichnis

- 1.) CAMAC A Modular Instrumentation System for Data Handling. Revised Description and Specification, ESONE Committee, EUR 4100e (1972)
- 2.) CAMAC Organisation of Multi-Crate Systems, ESONE Committee, EUR 4600 e (1972)
- 3.) J.G. Ottens: CAMAC Ein System rechnergeführter Elektronik, Elektronik 1970, H. 10, S. 335, H. 11, S. 387
- 4.) J.G. Ottens: CAMAC Ein System rechnergeführter Elektronik. Der vertikale Datenweg. Elektronik 1971, H. 2, S. 53
- 5.) CAMAC Bulletin, Zeitschrift für CAMAC-Anwendungen. Herausgegeben vom ESONE-Komitee
- 6.) W. Egl: CAMAC CRATE CONTROL für PDP 8 L, Österreichische Studiengesellschaft für Atomenergie
- 7.) T. Schlurick, H.-J. Trebst: Der Anschluß eines CAMAC-Crate-Kontrollers an die PDP 7, Jahresbericht Erlangen (1970/71) S. 188
- 8.) K.H. Timmesfeld et al: PEARL A proposal for a process- and experiment automation realtime language, KFK-PDV 1 Gesellschaft für Kernforschung mbH. Karlsruhe (1973)
- 9.) Official definition of CORAL 66, Inter-Establishment Committee for Computer Applications, Feb. 1970
- 10.) PAS 1 Prozeß-Automatensprache 1, BBC, Mannheim
- 11.) ASME-PEARL-Subset/1, ASME-Bericht, ESG München (1975)
- 12.) F. May, H. Halling, Petreczek, FOCAL Overlay for CAMAC Data and Command Handling CAMAC Bull., Nr. 1, (1971), S. 18
- 13.) H. Halling, K. Zwoll, W. John CAMAC Overlay for Single-User BASIC and Modification of 8-User BASIC for the PDP 11, CAMAC Bull., Nr. 6, (1973), S. 15
- 14.) Proposal for a CAMAC-Language, ESONE Committee, Software Working Group, CAMAC Bulletin Nr. 5 (1972)
- 15.) CAMAC, the Definition of IML, a Language for Use in CAMAC Systems, ESONE/IML/01, Okt. 1974

- 16.) PDP 15 FORTRAN IV Programmer's Reference Manual,
DEC-15-KFZB-D (1970)
- 17.) PDP 15 FORTRAN IV Operating Enviroment
DEC-15-GFZA-D (1971)
- 18.) F.R.A. Hopgood: Compiler Die Übersetzung von
Programmiersprachen, Carl Hauser Verlag München (1970)
- 19.) J.A. Lee: The Anatomy of a Compiler,
Van Norstrand Reinhold Company, New York (1967)
- 20.) C. Kuck: Programmsysteme für Realzeitrechner,
R. Oldenbourg Verlag München Wien (1972)
- 21.) VDI, Funktionelle Beschreibung von Prozeßrechner-
Betriebssystemen VDI/VDE 3554
Entwurf (1976)
- 22.) Programmed Data Processor-7 Users Handbook,
DEC, (1965)
- 23.) Advanced Monitor Software System für PDP 15/20/30/40,
DEC-15-MRZB-D (1970)
- 24.) RSX PLUS Real Time Executive Reference Manual,
DEC-15-RSXA-A-D (1972)
- 25.) T. Schlurick: Ein CAMACorientiertes Analysier-
und Auswertesystem (CoAAS)
Jahresbericht Erlangen (1973/74), S. 347
- 26.) P. Müller, T. Schlurick, H.-J. Trebst: Rechnergesteuerte
Aufnahme analoger Daten mit Hilfe von CAMAC,
Jahresbericht Erlangen (1973/74), S. 349
- 27.) T. Schlurick: Ein Programmsystem für die Impulshöhen-
analyse mit Spektrumsstabilisation, Diplomarbeit,
Erlangen 1968
- 28.) W. Latzel: Entwicklung eines Programmsystems für Echtzeit-
analyse zusammen mit einer geeigneten Modifikation des
Betriebssystems der PDP 7, Diplomarbeit, Erlangen 1974
- 29.) Parnas: Vorläufige Fassung eines Skriptums über
Betriebssysteme
- 30.) H.J. Schneider: Compiler Aufbau und Arbeitsweise,
De Gruyter Lehrbuch, Berlin-New York (1975)

Herrn Prof. Dr. N. Fiebiger danke ich für die Überlassung der Arbeit, für sein stetes, förderndes Interesse, sowie für die kritische Durchsicht der Arbeit.

Herrn Prof. Dr. F. Hofmann danke ich für wertvolle Hinweise zur Darstellung der Arbeit.

Herrn Dr. H.-J. Trebst fühle ich mich verbunden für viele interessante Diskussionen über CAMAC.

Bei Herrn Dipl.-Phys. Paul Müller bedanke ich mich für den Einsatz des Programmsystems im Experiment und für die Überlassung der Bilder.

Anhang A: Programme zur Messung von T_C -Änderungen

TASK 1

```
C      INITIALISIERUNGSTASK
C
C      LOESCHEN ALLER VERWENDETEN DATEN BEREICHE
      Z 1
      Z 2
      Z 3
      Z 4
      Z 5
      Z 6
C      FLAG FÜR WECHSELPUFFERBETRIEB
      IW=0
C      SYNCHRONISATIONSFLAG
      IFLGS=0
      IFLGD=0
C      LAUFINDEX FÜR DATENABSPICHERUNG
      K=1
103    FORMAT(6H START)
      WRITE(4,103)
C      SETZEN DER VORWAHLZEIT IM VORWAHLREGISTER
      WRITEC 6:0G2,I TIME
C      LOESCHE LAM
      CLEAR 6:0
C      SCHALTE SIEMENSVOLTMETER EIN
      ENABLE 2:12
C      SCHALTE VORWAHLZAEHLER EIN
      ENABLE 6:0
      END
```

TASK 2

```
C      STARTE MESSUNG, ANGESTOSSEN DURCH EIN LAM VOM VORWAHLZAEHLER
C
C      LOESCHE LAM VOM VORWAHLZAEHLER
      CLEARL 6:0
C      STARTE SIEMENSVOLTMETER
      CLEAR 2:0
C      STARTE DANAVOLTMETER
      ENABLE 3:0
      ENABLE 4:0
      END
C
```

TASK 4

```

C      TASK ZUM AUSLESEN DES DANAVOLTMETERS
C
C      SCHALTE EINGABEREGISTER AUS
      DISABL 3;0
      DISABL 4;0
C
C      LIES MESSWERT
      RDCBCD 3;0,MW1
C
C      LIES BEREICHSEINSTELLUNG
      READCL 4;0,MW2
C
C      UEBERGABE DER TASK ANS BETRIEBSSYSTEM
      TAKE
C
C      WECHSELPUFFERFLAG
      IF IW 10,11,10
C
C      ABSPEICHERUNG IN DIE ZWEITEN PUFFER
10      IB4(K)=MW1
      IB5(K)=MW2
      GOTO 12
C
C      ABSPEICHERUNG IN DIE ERSTEN PUFFER
11      IB1(K)=MW1
      IB2(K)=MW2
C
C      IST SIEMENS FERTIG
12      IF IFLGS 31,31,30
C
C      NEIN, DANN SETZE DANAFLAG FERTIG
31      IFLGD=1
      RETURN
C
C      JA, DANN LOESCHE FLAGS
30      IFLGS=0
      IFLGD=0
C
C      IST PUFFERENDE ERREICHT
      IEND=K-1024
      IF IEND 41,51,51
C
C      JA, UMSCHALTUNG AUF NEUE PUFFER
51      K=0
      IF IW 52,53,52
53      IW=1
101     FORMAT(6H ENDE1)
      WRITE(4,101)
C
C      LOESCHEN DER NEUEN PUFFER
      Z 4
      Z 5
      Z 6
C
C      ERHOEHE ABSPEICHERINDEX
41      K=K+1
C
C      START DES VORWAHLZAHLERS
      CLEAR 6;0
      RETURN
52      IW=0
102     FORMAT(6H ENDE2)
      WRITE(4,102)
C
C      LOESCHEN DER NEUEN PUFFER
      Z 1
      Z 2
      Z 3
      GOTO 41
      END
C

```

TASK 5

```

C      AUSLESEN DES SIEMENS-VOLTMETERS
C
      READC 2:0,MW3
C      UEBERGABE ANS BETRIEBSSYSTEM
      TAKE
C      AUSBLENDEN DER BEREICHSANZEIGE
      N=MW3-1
      KN=MW3-64/N
      KN=MW3-2
      KN=KN-KC
C      BERUECKSICHTIGUNG DER BEREICHSANZEIGE
3      IF KN 5,5,4
4      N=N*10
      KN=KN-1
      GOTO 3
C      WECHSELPUFFERFLAG
5      IF IW 72,73,72
C      ABSPEICHERUNG IN DIE ZWEITEN PUFFER
72     IB6(K)=N
      GOTO 74
C      ABSPEICHERUNG IN DIE ERSTEN PUFFER
73     IB1(K)=N
C      IST DATA FERTIG
74     IF IFLGD 6,6,7
C      NEIN, DANN SETZE SIEMENS FERTIG
6      IFLGS=1
      RETURN
C      JA, DANN LOESCHE FLAGS
7      IFLGD=0
      IFLGS=0
C      IST PUFFERENDE ERREICHT
      IEND=I-1024
C      ERHOEHEN ABSPEICHERINDEX
      K=K+1
      IF KEND 40,50,50
C      JA, UMSCHALTUNG AUF NEUE PUFFER
50     K=1
      IF IW 75,76,75
76     IW=1
      WRITE(4,101)
C      LOESCHEN DER NEUEN PUFFER
      Z 4
      Z 5
      Z 6
C      START DES VORHAELZAEHLERS
40     CLEAR 6,0
      RETURN
75     WRITE(4,102)
      IW=0
C      LOESCHEN DER NEUEN PUFFER
      Z 1
      Z 2
      Z 3
      GOTO 40
      END
C

```

TASK 7

```
C      TASK ZUM ABSPEICHERN VON DREI PUFFERN AUF DECTAPE
      WD 1
      WD 2
      WD 3
104    FORMAT(5H ENDE)
      WRITE(4,104)
      END
```

C

TASK 8

```
C      TASK ZUM ABSPEICHERN VON DREI PUFFERN AUF DECTAPE
C
      WD 4
      WD 5
      WD 6
      WRITE(4,104)
      END
```

Anhang B: Programm zum Übertrag von Spektren

```
C
C      STARTTASK ZUM UEBERTRAG VON SPEKTREN
C      VOM CANALIZER-VOL-1024 ANALYZER-UP
C
TASK 10
C      K IST KANALZAHL
C      K=1
C      I 1ST DIFFERENTIAL
C      I=1
C      SPEKTRUMSLAENGE
C      LAENGE=1024
C      ICB0=20,0
C      ICB1=20,1.
C      ICB0G0=00,000
C      KANALINHALT
C      LK=0
C      LOESCHE UEBERTRAGUNGSREGISTER
C      READC ICB1 F.
C      SETZE UEBERTRAGUNGSMODE (LAM NACH EINGABE)
C      WRITEC ICB0G2,0
C      ENABLE ICB1
C      END
```

```

0      TASK ZUM LESEN EINES ZEICHENS NACH EINEM LAM
0
0      READC ICR1,KZ
0      PRUEFEN AUF SONDERZEICHEN
0      N=KZ-$215
0      CARRIAGE RETURN (ABSPEICHERN)
0      IF N 1,7,1
0      N=KZ-$12
0      LINE FEED (UEBERLESEN)
0      IF N 2,11,2
0      N=KZ-$240
0      ZWISCHENRAUM (ABSPEICHERN)
0      IF N 3,7,3
0      N=KZ-$377
0      RUB-OUT (UEBERLESEN)
0      IF N 4,11,4
0      N=KZ-$173
0      ENDEZEICHEN
0      IF N 5,13,5
0      ZIFFER VERARBEITEN
0      IF KZ 6,11,6
0      N=KZ-$260
0      IF N 60,61,61
0      N=N+$200
0      LK=10*LK+N
0      I=I+1
0      RETURN
0      ZAHL DER GELESSENEN ZIFFERN
0      N=I-7
0      IF N 8,9,10
0      N=I-5
0      WAR ES EINE KANALZAHL
0      IF N 10,12,10
0      KANALINHALT ABSPEICHERN
0      IF K 10,10,14
0      N=K-LAENGE
0      IF N 15,15,10
0      IB1(K)=LK
0      K=K+1
0      RUECKSPRUNG BEI FEHLER
0      LK=0
0      I=1
0      RETURN
0      GELESENE KANALZAHL SETZEN
0      K=LK
0      RETURN
0      ENDE DER UEEERTRAGUNG
0      DISABL ICB0
0      FORMAT (5H ENDE, I6, 3H KANAELE)
0      K=K-1
0      WRITE(4,89)K
0      END

```

Anhang C:

```
/
/      CAMAC MACROS FUER DEN CCC2-CONTROLLER AN DER PDP 7
/
/      T. SCHLURICK, 25. 10. 74
/
/      .GLOBL CCA
/
/      .DEFIN INITC
JMS* CCA
1
. ENDM
/
/      .DEFIN CLOSEC
2
. ENDM
/
/      .DEFIN INIT
JMS* CCA
3
2
. ENDM
/
/      .DEFIN INHIBT
JMS* CCA
3
3
. ENDM
/
/      .DEFIN CLRINH
JMS* CCA
3
4
. ENDM
/
/      .DEFIN ZEROAL
JMS* CCA
3
1
. ENDM
/
/      .DEFIN CLEAR,NA
JMS* CCA
3
220
NA
. ENDM
```

/
.DEFIN READCL, NA, K
JMS* CCA
10
NA
46
K
.ENDM

/
.DEFIN READCM, NA, K
JMS* CCA
10
NA
66
K
.ENDM

/
.DEFIN RDBCD, NA, K
JMS* CCA
10
NA
2006
K
.ENDM

/
.DEFIN RDCBCD, NA, K
JMS* CCA
10
NA
3006
K
.ENDM

/
.DEFIN WRITEC, NA, K
JMS* CCA
11
407
NA
K
.ENDM

/
.DEFIN WRITES, NA, K
JMS* CCA
11
447
NA
K
.ENDM

/
.DEFIN TRANS, NA1, NA2
JMS* CCA
4
1
NA1
NA2
.ENDM

./
.DEFIN CLEARL,NA
JMS* CCA
3
240
NA
.ENDM

./
.DEFIN ENABLE,NA
JMS* CCA
3
540
NA
.ENDM

./
.DEFIN DISABL,NA
JMS* CCA
3
500
NA
.ENDM

./
.DEFIN EXECUT,NA
JMS* CCA
3
520
NA
.ENDM

./
.DEFIN TSTSTA,NA
JMS* CCA
3
560
NA
.ENDM

./
.DEFIN TSTLAM,NA
JMS* CCA
3
560
NA
.ENDM

./
.DEFIN READC,NA,K
JMS* CCA
10
NA
6
K
.ENDM

Betriebsanleitung

Das CoAAS steht als Benutzerprogramm sowohl für die PDP 7 als auch für die PDP 15 zur Verfügung. Dadurch ist es möglich, Programme an der PDP 15 zu entwickeln und syntaktisch zu prüfen. Soweit Tasks keine CAMAC-Ein-Ausgaben enthalten, können sie auch auf der PDP 15 laufen. Diese Möglichkeit kann auch zur Auswertung von Meßdaten ausgenutzt werden.

Das System ist in der Quell- und in der Binärfassung auf einem DECTape abgespeichert. Als Hauptprogramme stehen die Programme AASN7 und AASN mit ihren beiden Librarys für die PDP 7 u. PDP 15 zur Verfügung. (Die PDP 15 benötigt andere Bildschirm- und Plotterprogramme, was in den Librarys berücksichtigt ist.)

Das DECTape muß unter UNIT1 aufgelegt werden. Auf der PDP 15 sollte es mit RFSAVE auf die Platte überspielt werden. An der PDP 7 muß das Systemband (16K) auf UNIT8 aufgelegt und danach mit dem Monitorloader geladen werden (Startadresse siehe LOADER).

Das CoAAS wird über den Systembefehl

\$GLOAD

geladen. Das System meldet sich mit

LOADER

Danach ist

AASN7 (für die PDP 7) bzw.

-AASN (für die PDP 15)

einzutippen, wodurch das CoAAS geladen wird. Während des Ladelaufs wird auf der PDP 7 die Ladefliste ausgegeben. Die Bereitschaft des Systems wird durch den Ausdruck

AASN

>

gezeigt.

Wenn Tasks eingegeben oder geladen werden sollen, muß ein DECTape mit der UNIT2 bei der PDP 7 aufgelegt werden, (das System arbeitet mit den logischen Kanalnummern 1 und 2)

das die Quellprogramme aufnimmt. Dazu muß das Laufwerk auf WRITE-ENABLED gestellt werden.

Von dem DECTape können auch Tasks geladen werden, die mit dem EDITOR bearbeitet wurden.

Verkürzte Ladeprozeduren

Weil die Ladeprozedur von den DECTapes langwierig ist, wird empfohlen, das geladene System auf dem Q-Bereich des Systembandes abzuspeichern. Dies erfolgt durch

CNTRL QØ

Dazu muß das Systemband auf WRITE ENABLED stehen. Danach kann das CoAAS jederzeit schnell über den Befehl

G Ø

geladen, und wenn das DECTape steht, mit

CNTRL S

gestartet werden. Es meldet sich wie bei der normalen Lade-prozedur mit

AASN

Die "Feuerwehrversion"

Diese Arbeitsweise ist auch für ein System mit geladenen Tasks zu empfehlen. Wenn die Programme ausgetestet sind, sollte die Version auf ein DECTape unter der UNIT3 abgespeichert werden. Dies erfolgt analog mit

CNTRL Q3

Auch hier muß das DECTapelaufwerk auf WRITE-ENABLED gestellt werden. Danach kann das System als "Feuerwehrversion" jederzeit wieder mit

G Ø

geladen und nach dem das DECTape 3 steht mit

CNTRL S

gestartet werden. Dabei werden die Datenbereiche nicht zerstört.

Anhang E: Aufbau des Compilers

In Kapitel 5 wurde die Struktur des Compilers beschrieben. Hier soll der Aufbau der Teilprogramme und deren Zusammenspiel näher erläutert werden.

1. Das Eingabeprogramm

Das Eingabeprogramm umfaßt die Bearbeitung einer Anweisung von der Texteingabe bis zur Erkennung des Schlüsselworts. Danach wird die Verarbeitung in der dazugehörigen Semantikroutine durchgeführt.

Während Einzelanweisungen nur über den Teletype eingegeben werden können, lassen sich Tasks auch vom DECTape aus übersetzen.

Bei der Eingabe über den Teletype wartet die Compilertask, die bei der Initialisierung des Systems automatisch gestartet wird, solange keine andere Task den Teletype benutzt, im Wartezustand auf eine Eingabe. Durch ein CR, mit der jede Eingabe abgeschlossen wird, wird die Compilertask fortgesetzt.

Bei der Eingabe vom DECTape wird durch die Anweisung LOAD das Lesen und Übersetzen einer kompletten Task gestartet. In diesem Fall wird sequentiell der Text eines Files vom Band gelesen und die Compilertask wird nach dem Lesen eines Strings fortgesetzt.

Für die ersten beiden Übersetzungsschritte, der Entschlüsselung einer möglichen Anweisungsnummer und der Schlüsselworterkennung, werden vom Eingabeprogramm das Syntaxanalysierprogramm benutzt.

Eine eventuell vorhandene Anweisungsnummer, die vom Syntaxanalysierprogramm als Zahl entschlüsselt wird, wird lediglich für die Abschlußsteuerung abgespeichert. Eine Bearbeitung im Eingabeprogramm erfolgt noch nicht, weil erst die Fehlerfreiheit der zu bearbeitenden Anweisung geprüft werden muß.

Der für die Übersetzung wichtigste Schritt beginnt mit der Bearbeitung des ersten Wortes einer Anweisung (Wort = 1. Zeichen eines Stringteils ein Buchstabe). Es wird vom Syntaxanalysierprogramm aufgebaut und danach mit der Schlüsselworttabelle verglichen. In dieser Tabelle sind die Schlüsselworte aller Anweisungen aufgeführt sowie die Adressen der dazugehörigen Semantikroutinen, so daß das Eingabeprogramm die Verarbeitung der Anweisung nach der Erkennung des Schlüsselwortes an die Semantikroutine übergeben kann.

Wenn das erste Wort der Anweisung nicht in der Schlüsselworttabelle aufgeführt ist, wird die weitere Bearbeitung der Arithmetikübersetzungsroutine übergeben.

Bei der Verwendung von Schlüsselworten als Variablenamen können Doppeldeutigkeiten auftreten. In diesem Fall entscheidet das nachfolgende Trennungszeichen über die Weiterverarbeitung, so daß bei einem Gleichheitszeichen die Anweisung auch der Arithmetikroutine übergeben wird.

2. Die Schlüsselworttabelle

Die Schlüsselworttabelle hat eine doppelte Funktion. Einmal legt sie durch die Anzahl der in der Liste aufgeführten Schlüsselworte den Sprachumfang des Compilers fest. Gleichzeitig werden auch in der Liste die Adressen für die verarbeitenden Semantikroutinen angegeben. Durch die Verwendung von globalen Adressen in der Schlüsselworttabelle wird erreicht, daß zur Ladezeit des Systems alle benötigten Semantikroutinen automatisch aus der Library geladen werden.

Es ist deshalb leicht möglich, durch Veränderung der Schlüsselworttabelle vor dem Ladevorgang sprachliche Varianten des Compilers herzustellen. Das hat eine besondere Bedeutung für Benutzer mit unterschiedlichen Anforderungen. Sie können über die Schlüsselworttabelle die für ihr Experiment benötigten Funktionen und Sprach-elemente zusammenstellen. Außerdem können neue Funktionen

mit entsprechenden Semantikroutinen in die Sprache eingefügt werden.

Ein Eintrag in der Schlüsselworttabelle besteht aus 3 Worten. In den ersten beiden Worten ist das Schlüsselwort im .SIXBT-Code eingetragen, was 6 Zeichen entspricht und im dritten Wort ist die Adresse eingefügt, die auch als Globale in einer .GLOBAL-Anweisung aufgeführt sein muß.

Da Schlüsselworttabelle und Semantikroutinen getrennt als Programmmodule übersetzt werden können, kann die Schlüsselworttabelle unabhängig von den Semantikroutinen bearbeitet werden.

Es ist außerdem möglich, verschiedene Schlüsselworttabellen für verschiedene Aufgaben vorzubereiten, die erst zur Ladezeit mit aufgerufen werden.

3. Die syntaktische Analyse

Das Programm zur syntaktischen Analyse steht dem Eingabeprogramm sowie allen Semantikroutinen als Hilfsprogramm zur Verfügung. Es bildet das Koppelglied zwischen dem Eingabestring und den Verarbeitungsroutinen. Der Begriff Analyse ist dabei als Zerlegung zu verstehen.

Das syntaktische Analysierprogramm wird nach jeder Eingabe eines Strings, genauer beim Start der Übersetzung eines jeden Strings, auf den Anfang des Strings eingestellt. Danach liefert es nach jedem Aufruf den bis zum nächsten Trennungszeichen liegenden Stringteil sowie das Trennungszeichen selbst.

Als Trennungszeichen gelten:

, () + - */

Außerdem gilt der Wechsel von einer Zahl zu einem Buchstaben als Trennung.

Gleichzeitig werden die Stringteile für eine spätere Listenverarbeitung umcodiert. Dazu werden Zahlen und Namen unterschieden.

Als Zahlen werden alle Stringteile übersetzt, die als erstes Zeichen entweder am Anfang eines Strings oder nach einem Trennungszeichen eine Ziffer enthalten. Solange kein $\$$ -Zeichen einer Zahl voransteht, wird die Zahl als Dezimalzahl aufgefaßt, und die Ziffern werden entsprechend in eine Dualzahl umgewandelt bis wieder ein Trennungszeichen oder ein Buchstabe folgt. Ein voranstehendes $\$$ -Zeichen interpretiert die nachfolgende Zahl als Oktalzahl. Im Syntaxanalysierprogramm wird neben der Umwandlung von Oktal- und Dezimalzahlen auch bereits die Umschlüsselung der CAMAC-Adressen von n und a in die interne Integerdarstellung durchgeführt.

Als Namen werden alle Stringstücke aufgefaßt, die nach einem Trennungszeichen oder am Stringanfang mit einem Buchstaben beginnen. Danach dürfen in beliebigem Wechsel Buchstaben und Ziffern folgen. Abgeschlossen werden Namen stets nur durch ein Trennungszeichen. Obwohl Namen beliebiger Länge eingegeben werden können, werden vom Syntaxanalysierprogramm nur 6 Zeichen als relevant gespeichert und in zwei Worten für die Weiterverarbeitung zur Verfügung gestellt. Außerdem wird untersucht, ob der erste Buchstabe des Namens ihn als Integer- oder Realzahlvariablenamen festlegt, was für die spätere Bearbeitung mit einer Flag angezeigt wird.

4. Die Semantikroutinen

Nach der Erkennung des ersten Schlüsselwortes im Eingabeprogramm wird die Übersetzung der einzelnen Anweisungen in den Semantikroutinen durchgeführt. Im allgemeinen hat jedes Schlüsselwort seine eigene Semantikroutine, wobei logisch zusammenhängende Routinen zu Programmmodulen zusammengefaßt sind. Die Semantikroutinen benutzen als Eingabeprogramm das Syntaxanalysierprogramm. Zur Bearbeitung von Namen und Zahlen stehen ein Listenmanipulationsprogramm mit Listen für Namen sowie Codeabsetzroutinen

und Platzreservierungsroutinen für Daten zur Verfügung. Aufgrund der umfangreichen Hilfsprogramme ist es leicht möglich, neue Semantikroutinen zu implementieren.

Die Anfangsadresse jeder Semantikroutine muß als Globale deklariert werden, die zusammen mit dem Schlüsselwort für die Routine in der Schlüsselworttabelle eingetragen werden muß. Jede Semantikroutine oder ein Modul mit mehreren Routinen kann getrennt übersetzt werden. Dazu sind ebenfalls alle benötigten Hilfsroutinen als Globale anzugeben. Die übersetzten Programmodule werden in der Programmlibrary zusammengefaßt, aus der sie nach der Zusammenstellung in der Schlüsselworttabelle geladen werden. Die Semantikroutinen übergeben nach der Übersetzung einer Anweisung die Steuerung an die Abschlußsteuerung.

Den Semantikroutinen gleichgeschaltet sind die Compileranweisungen. Im Gegensatz zu den Semantikroutinen liefern sie keinen übersetzten Code, sie werden sofort ausgeführt. Außerdem benötigen sie nicht die Abschlußsteuerung nach der erfolgten Ausführung, sondern sie übergeben die Steuerung sofort wieder dem Eingabeprogramm.

5. Die Listенbearbeitungsroutinen und Codeabsetzroutinen

Im CoAAS werden die Namen von einfachen Variablen indirekt und von dimensionierten Variablen explizit definiert. Zur Wiederauffindung der reservierten Plätze und zum Eintrag dieser Variablen werden Listen geführt, die mit zwei Listенbearbeitungsroutinen manipuliert werden. Um Speicherplatz zu sparen, werden außerdem Konstante unter einer speziellen Kennzeichnung in der Namensliste abgespeichert, so daß bei mehrfachem Gebrauch einer Konstanten diese stets nur einmal abgespeichert wird. Außerdem sind auch die Anweisungsnummern in den Listen enthalten.

Von den beiden Listенbearbeitungsroutinen wird die erste für das Aufsuchen der Namen benötigt. Dazu muß die Routine mit dem gesuchten Namen, der Listenadresse und einem Zeiger auf die Listenlänge versorgt werden. Zwei Ausgänge der Routine zeigen an, ob der gesuchte Name bereits in der Liste gefunden wurde oder nicht. Weil jeweils die Listenadresse angegeben werden muß, kann die Routine zur Bearbeitung mehrerer Listen eingesetzt werden, so z.B. auch für die Schlüsselworterkennung.

Die zweite Routine ergänzt die bearbeitete Liste und kann deshalb erst nach der Suchroutine verwendet werden, wenn ein Name nicht vorhanden ist. Namen werden in 2 Worten im .SIXBT-Code eingetragen. Eine weitere Zelle enthält die Adresse des reservierten Speicherplatzes. Außerdem sind in 3 Bit Kennzeichen über die Art des Namens von einfacher oder dimensionierten Integer- oder Realzahl-namen enthalten. Mit den Routinen lassen sich auch andere Informationen als Adressen bearbeiten.

Der von den Semantikroutinen erzeugte Binärcode wird in einem Zwischenpuffer gespeichert, bis jeweils eine Anweisung vollständig übersetzt ist. Zum Absetzen des Binär-codes stehen spezielle Routinen zur Verfügung. Sie arbeiten zum Teil in Verbindung mit dem Syntaxanalysier- und dem Listenverarbeitungsprogramm, so z.B. wenn Argumente von Funktionen aus einer Anweisung entschlüsselt und als Adressen im Binärcode abgesetzt wurden.

6. Die Übersetzung arithmetischer Ausdrücke

Die arithmetischen Ausdrücke werden in einem direkten Verfahren, d.h. ohne Verwendung einer Zwischennotation, übersetzt. Damit die Übersetzungsroutine nicht nur für Zuweisungen benutzt werden kann, beginnt die eigentliche Übersetzung eines arithmetischen Ausdrucks erst nach dem Zuweisungszeichen.

Bei der Übersetzung spielen zwei Begriffe eine wesentliche Rolle:

1. Die Operatorklasse

Die Operationen werden in Operatorklassen eingeteilt. Zu den Operatoren der vier Grundrechnungsarten wird noch der Zeilenabschluß ($\downarrow$) als Operator zur Kennzeichnung der Übersetzungsbeendigung hinzugenommen, so daß damit folgende zwei Klassen gebildet werden:

$$\begin{aligned} \langle \text{Operatorklasse } \emptyset \rangle &::= + \quad - \quad \downarrow \\ \langle \text{Operatorklasse } 1 \rangle &::= * \quad / \end{aligned}$$

2. Die Kellerebene

Außer der Operatorklasse ist die Kellerebene von Bedeutung, die aus der Klammerung errechnet wird. Eine linke Klammer erniedrigt die Kellerebene um 1, während eine rechte Klammer die Kellerebene um 1 erhöht.

6.1 Die Behandlung eines Terms in der Kellerebene

Die Übersetzung erfolgt zunächst nach Kellerebenen getrennt, d.h. sie startet mit jeder Kellerebene neu. Erzielte Zwischenergebnisse werden beim Übergang zur höheren Kellerebene berücksichtigt. Deshalb soll zuerst die Übersetzung in einer Kellerebene betrachtet werden, d.h. die Übersetzung eines Ausdrucks ohne Klammern.

Jedem Ausdruck wird, falls er kein explizites Vorzeichen trägt, automatisch ein positives Vorzeichen als erster Operator vorangestellt. Danach werden nacheinander die Operatorklassendifferenzen der begrenzenden Operatoren eines jeden Elements gebildet. Entsprechend der Definition der Operatorenklassen kann die Differenz +1, $\emptyset$ oder -1 betragen.

Beträgt die Operatorenklassendifferenz $\emptyset$, so kann die vor dem Argument stehende Operation mit ihm sofort als Binärteil übersetzt werden.

Das folgende Beispiel zeigt einen einfachen Ausdruck und das übersetzte Programm:

Ausdruck:	+ a - b + c
Operatorklasse:	$\emptyset$ $\emptyset$ $\emptyset$ $\emptyset$
Operatorklassendifferenz:	$\emptyset$ $\emptyset$ $\emptyset$

Daraus folgt das Programm

Lösche	AC	} Lade A
Addiere	A	
Subtrahiere	B	
Addiere	C	

Dabei werden die Operationen Lösche und Addiere zu einer Operation Lade zusammengefaßt.

Beim Auftreten der Operatorklassendifferenz -1 muß für das bis dahin übersetzte Programmstück ein Zwischenergebnis mit dem dazugehörigen Vorzeichen abgelegt werden. Das entspricht der Aufspaltung eines Ausdrucks in zwei Teilausdrücke, wie das folgende Beispiel zeigt:

Ausdruck:	+ a - b + c / d * g / h
Operatorklassendifferenz:	$\emptyset$ $\emptyset$ -1 $\emptyset$ $\emptyset$ +1

Dieser Ausdruck geht über in:

$$+ a - b = +a' \text{ und } + c / d * g / h$$

Das Zwischenergebnis a' wird nach dem Auftreten der Operatorklassendifferenz +1 mit berücksichtigt, so daß folgender Ausdruck entsteht:

$$+ c / d * g / h + a'$$

Die Adresse für das Zwischenergebnis a' sowie der Operator vor a' müssen dazu zwischengespeichert werden.

Zwei Sonderfälle werden noch berücksichtigt:

1. Die Operatorklassendifferenz -1 tritt beim ersten Argument auf. In diesem Fall braucht kein Zwischenergebnis berechnet und eine Adresse zwischengespeichert

zu werden. Diese Situation wird gekennzeichnet, so daß auch später ein Zwischenergebnis nicht berücksichtigt wird.

2. Nach der ersten Operatorklassendifferenz $\emptyset$ folgt sofort die Differenz -1. In diesem Fall wird die Adresse des ersten Arguments als Adresse des Zwischenergebnisses abgespeichert. Außerdem muß der Operator abgespeichert werden.

Da bei den Operatorklassendifferenzen maximal 1 auftreten kann, reicht ein Zwischenspeicher (ZS) für die Aufnahme der Adresse und des Operators für das Zwischenergebnis aus.

6.2 Der Übergang zwischen Kellerebenen

Jeder arithmetische Ausdruck startet mit der Übersetzung bei der Kellerebene $\emptyset$. Deshalb muß die Übersetzung, wenn der Ausdruck korrekt ist, wieder bei der Ebene $\emptyset$ enden und außerdem darf keine positive Kellerebene auftreten.

Genau wie beim Auftreten einer negativen Operatorklassendifferenz muß beim Übergang zu einer niederen Kellerebene ein Zwischenergebnis gebildet und die Adresse einschließlich Vorzeichen abgespeichert werden. Da im Gegensatz zu Operatorklassendifferenzen niedrigere Kellerebenen als -1 erreicht werden können, ist ein Zwischenspeicher nicht mehr ausreichend, sondern die Adressen für die Zwischenergebnisse müssen gekellert werden. Auf jeder Kellerebene müssen maximal zwei Adressen mit ihren Operatoren auf zwei Kellerplätzen sowie der Operator vor dem Klammerausdruck gespeichert werden.

Auf dem ersten Kellerplatz einer Kellerebene muß die Adresse eines Zwischenergebnisses sowie das Vorzeichen aus dem ZS gerettet werden, da auf der neuen Kellerebene der ZS wieder benutzt werden kann. Als zweite Adresse muß die Adresse des bis zum Auftreten der linken Klammer gebildeten

Zwischenergebnisses mit den dazugehörigen Vorzeichen abgespeichert, während auf dem dritten Speicherplatz der Operator vor der Klammer gerettet wird. Danach startet die Übersetzung auf der niederen Kellerebene neu.

Der Übergang zur höheren Kellerebene erfolgt in drei Schritten:

1. Zuerst wird der ZS der aktuellen Kellerebene abgearbeitet. Das bedeutet, daß ein eventuelles Zwischenergebnis, dessen Adresse im ZS gehalten wird, berücksichtigt wird, bevor auf die nächst höhere Kellerebene zurückgegangen wird.
2. Danach wird der ZS aus dem ersten Kellerplatz des Kellerspeichers für die höhere Kellerebene restauriert.
3. Mit dem 3. Schritt wird die Übersetzung auf der höheren Kellerebene fortgesetzt, indem die Operatorklassendifferenz zwischen den Operatoren vor und nach der Klammer gebildet wird. Weil das Ergebnis des Klammersausdrucks, der das 2. Argument darstellt, bereits im AC steht, müssen für die Subtraktion und die Division Unterprogramme für vertauschte Argumente eingesetzt werden, wobei das erste Argument aus dem Kellerspeicher geholt wird.

Im folgenden Beispiel wird die Übersetzung eines einfachen Klammersausdrucks in tabellarischer Form in 7 Schritten gezeigt:

Ausdruck:	+a	+b	*	(	+c	-d	)	+e	↓
Operatorklassendifferenz (OKD)	0	-1			0	0		+1	0
Kellerebene (KE)	0	0	-1	-1	-1	0	0		
Schritt	1.	2.	3.	4.	5.	6.	7.		

Der vorliegende Ausdruck erzeugt aufgrund der Operatorklassendifferenzen und Kellerebenenübergänge folgendes Programm:

Schritt	OKD	KE	Übersetzungsreaktion	Programm
1.	0	0	+a	
2.	-1	0	+a ZS	
3.		-1	ZS 1. Kellerplatz +b 2. Kellerplatz * 3. Kellerplatz (der 0. Kellerebene)	
4.	0	-1	+c	LADE c
5.	0	-1	-d	SUB d
6.	+1	0	ZS ist leer ZS wird restauriert (+a) *b ZS berücksichtigen	MUL b ADD a
7.	0	0	+e	ADD e

7. Die Abschlußsteuerung

Die Abschlußsteuerung bearbeitet 3 Aufgaben:

1. Sie kontrolliert das Abschlußzeichen des eingegebenen Strings. Von den Semantikroutinen braucht deshalb ein String nur bis zur logischen Erkennung der vollständigen Anweisung bearbeitet zu werden, so daß der Binärkode abgesetzt werden kann. Eventuelle nachfolgende Elemente, die syntaktisch nicht mehr erlaubt sind, werden von dem Abschlußprogramm als Fehler erkannt, und eine solche Anweisung wird verworfen.
2. Danach wird der Übertrag vom Zwischenpuffer in den Programmpuffer vorgenommen. Nach der Erkennung des Stringabschlußzeichens steht die Fehlerfreiheit der

Anweisung fest, so daß eine eventuelle vom Eingabeprogramm abgespeicherte Anweisungsnummer eingesetzt werden kann, und das Binärprogramm vom Zwischenspeicher in den Programmpuffer übertragen wird.

3. Der letzte Schritt hängt vom Zustand des Compilers ab. Wenn er im direkten Ausführungsmodus ist, wird die letzte im Programmpuffer abgespeicherte Anweisung sofort angesprungen und ausgeführt. Außerdem bleibt der Programmpufferzähler für den Programmpuffer unverändert. Dadurch überspeichert die nächste Anweisung dieselbe Stelle im Programmpuffer.

Im Taskübersetzungsmodus wird der Programmzähler um die Länge des im Programmspeicher abgesetzten Codes erhöht, so daß der Code der nächsten Anweisung anschließend abgespeichert wird. Danach steuert das Abschlußprogramm je nach dem Eingabegerät das Eingabeprogramm für den Teletype oder für das DECTape an.

8. Die Organisation der Anweisungsnummern

Besonderes Augenmerk mußte auf die Organisation der Anweisungsnummern gelegt werden, weil während der Übersetzung noch nicht alle Adressen bekannt sind.

Die Anweisungsnummern werden für drei verschiedene Aufgaben eingesetzt:

1. Zur Adressierung von FORMAT-Anweisungen,
2. als Sprungadresse für Sprunganweisungen und
3. als Adresse für das Ende einer Laufanweisung.

Da die FORMAT-Anweisungen nicht in den Programmpuffer, sondern in den Datenpuffer übernommen werden, werden die Anweisungsnummern der FORMAT-Anweisungen intern unterschiedlich gegenüber den Anweisungsnummern von Sprunganweisungen dargestellt. Damit wird auch leicht verhindert, daß die Adresse einer FORMAT-Anweisung als Sprungadresse benutzt wird. Es ist dadurch aber möglich, die gleiche

Anweisungsnummer sowohl vor einer FORMAT-Anweisung als auch vor einer Ausführungsanweisung zu verwenden.

Durch die Vorschrift, daß FORMATE bereits vor dem Aufruf in einer Ein-Ausgabeanweisung definiert sein müssen, brauchen für diese keine weiteren Sicherheitsvorkehrungen getroffen zu werden.

Andere Anweisungsnummern erhalten einen Sprungkontrollblock SB. In diesem Sprungkontrollblock werden die Kernspeicheranfangs- und -endadressen der übersetzten Anweisung geführt, sowie ein Pointer auf einen möglichen Loopblock LB, so daß damit im Datenpuffer 3 Zellen dafür benötigt werden (vergl. Abb. 1).

Anfangsadresse
Pointer auf Loopblock (oder 0)
Endadresse

Abb. 1: Sprungkontrollblock SB.

Alle Programmsprünge werden indirekt über die Sprungadressen in den Kontrollblöcken durchgeführt. Solange die Kernspeicheradressen noch nicht definiert sind, d.h. die Anweisungsnummer nur als Zieladresse in einer Anweisung aufgerufen wurde, ist als Adresse eine Fehlerausgangsroutine eingesetzt, die eine Task beendet, die auf eine undefinierte Adresse stößt. Damit wird ein Fehler in der Programmierung zur Laufzeit abgefangen.

Für die Organisation von Laufanweisungen ist ein Sprungkontrollblock nicht ausreichend, da mehrere Loop's auf einer Anweisungsnummer enden können. Deshalb wird für jede Laufanweisung ein Loopblock LB angelegt, der aus 5 Zellen besteht. Darin werden drei Adressen sowie eine Verzeigerung auf den zugehörigen Sprungblock SP geführt.

Wie bei direkten Sprunganweisungen werden auch die Sprünge in einem Loop indirekt ausgeführt.

Die drei Adressen werden

1. für den Einsprung in den Loop
2. für die Wiederholung des Loops am Ende aller Anweisungen im Loop und
3. für den Ausgang aus dem Loop nach erfüllter Loopbedingung benötigt.

Während die ersten beiden Adressen von der Bearbeitungs-routine der DO-Anweisung eingetragen werden, kann die letzte Adresse erst ergänzt werden, wenn die Anweisungs-nummer des Loopendes definiert ist. Deshalb ist eine Ver-zeigerung im Sprungblock notwendig, die einen Hinweis auf einen bzw. mehrere Loopenden enthält. Die Verzeigerung ist so organisiert, daß der Zeiger des Sprungblocks auf die innerste Schleife zeigt und der Zeiger im folgenden Loopblock stets auf die nächst höhere Schleife hinweist. Das Ende wird durch eine 0 angezeigt. Zusätzlich ist auch eine Rückwärtsverzeigerung aufgenommen, um später auch Programmkorrekturen durchführen zu können.

Loopwiederholungsadresse
Loopeinsprungsadresse
vorhergehender LB oder SB
nächster LB
Loopendeadresse

Abb. 2: Der Loopblock LB

Zusammenstellung von englischen Begriffen, von denen es zwar deutsche Übersetzungen gibt, die sich aber so in die Laborsprache eingebürgert haben, daß die Verwendung der deutschen Übersetzungen Unklarheiten bringen würde.

Call-Handler	Generelle Betriebssystembearbeitungsroutine für die PDP 7/15
CAMAC-Crate	CAMAC-Überrahmen
CAMAC-Driver	CAMAC-Treiber
CAMAC-Kontroller	Steuereinheit für ein Crate
CAMAC-Modul	CAMAC-Einschub
CPU-Central-Processor Unit	Zentraleinheit
DECTape	Spezialmagnetband der Firma DEC
Directory	Adreßbuch
Devicedriver	Gerätetreiber
Device-Handler	Umfangreicher Gerätetreiber für die PDP 7/15
Interface	elektron. Geräteanpassungsschaltung
Interrupt	Unterbrechungssignal
IOT-Befehl	Maschinenbefehl für Ein-Ausgabe
Taskkontroller	Prozeßumschalter

Lebenslauf

Ich wurde am 27.1.1940 als Sohn des Konstrukteurs Werner Schlurick und seiner Ehefrau Charlotte, geborene Schwarz, in Halle/Saale geboren. Von 1946 bis 1954 besuchte ich die Grundschule und wechselte danach in eine naturwissenschaftliche Klasse der Oberschule über. 1958 legte ich an der Oberschule Ammendorf in Halle die Reifeprüfung ab.

Da meine Bewerbung zum Physikstudium an der Universität Halle mit der Begründung abgelehnt wurde, ich solle in die Produktion gehen, entschloß ich mich, den Beruf des Landvermessers zu erlernen. Die Lehre beendete ich 1960 mit der Facharbeiterprüfung beim Topographischen Dienst Schwerin.

Auf eine neuerliche Bewerbung 1960 zum Physikstudium an der Technischen Hochschule Dresden erhielt ich keine Antwort. Stattdessen bemühten sich meine Vorgesetzten vom Topographischen Dienst Schwerin, mich für den Eintritt in die Volksarmee zu bewegen. Da ich dies nicht mit meinem Gewissen vereinbaren konnte, verließ ich im Oktober 1960 die DDR nach Berlin (West).

In der Bundesrepublik arbeitete ich von Ende 1960 bis Anfang 1962 als technischer Rechner und Programmierer bei der Siemens AG. in Erlangen.

Im April 1962 legte ich an der Oberrealschule an der Löbleinstraße in Nürnberg eine Ergänzungsprüfung zu meiner Reifeprüfung ab und begann im SS 62 mit dem Physikstudium in Erlangen. Die Diplomhauptprüfung legte ich im Mai 1969 ab.

Danach war ich 4 Monate bei der Siemens AG. in Erlangen tätig, und seit Oktober 1969 bin ich wissenschaftlicher Angestellter am Physikalischen Institut der Universität Erlangen.

Im März 1963 heiratete ich die Sekretärin Christa Müller. Ich habe zwei Kinder, ein Mädchen und einen Jungen, die im Mai 64 und im August 68 geboren wurden.

Copy-Center 2000

852 ERLANGEN · Obere Karlstr. 30
Tel. 09131-26197